Spis treści

1. Wstęp

1.1. Problematyka i zakres pracy

Niniejsza praca dotyczy zakresu inżynierii oprogramowania, a w szczególności aplikacji przeznaczonej na urządzenia mobilne. Jej działanie jest dodatkowo wspierane przez webową aplikację serwerową, która ma dostęp do bazy danych.

W ostatnich latach nastąpił intensywny rozwój technologiczny telefonów komórkowych. Wzrost wydajności oraz umieszczanie w nich dodatkowych modułów sprawiły, że urządzenia mobilne zaczęły być wykorzystywane do celów innych niż komunikacja. Jednym z nich jest wspieranie różnego rodzaju aktywności sportowych, między innymi biegania. Aplikacje mogą wykorzystywać otrzymywane poprzez protokół bluetooth dane z mierników tętna lub aktualną lokalizację użytkownika pobieraną z wbudowanego w urządzenie modułu lokalizacji.

W momencie pisania niniejszej pracy na rynku znajduje się wiele takich aplikacji, a ich funkcjonalność opiera się głównie na analizie odbytych treningów. Użytkownik przeważnie ma możliwość wyświetlenia przebiegu pokonanej trasy na mapie, sprawdzenia ile czasu zajął bieg, jaki był całkowity przebyty dystans, a także porównania statystyk z poszczególnych fragmentów trasy. Niektóre z nich oferują także pewne funkcje społecznościowe. Możliwe jest zapisywanie przebiegu pokonanej trasy, a następnie udostępnienie jej. W wyniku tego działania inni biegacze mają możliwość wyszukania trasy i odbycia na niej treningu we własnym zakresie. W przypadku niektórych aplikacji użytkownicy mają dodatkowo możliwość porównania swojej próby na konkretnej trasie z próbami innych zawodników na podstawie całkowitego czasu treningu. Aplikacje zawierające wymienione powyższe funkcje mają jednak pewne braki i niedoskonałości.

Po pierwsze, zawody biegowe różnią się od siebie dystansem i rodzajem terenu - niektóre wiodą przez trasę o twardej nawierzchni (na przykład asfalt, kostka brukowa) i wyrównanym poziomie, zaś inne przez nieutwardzone grunty wymagające podbiegów i zbiegów. Choć na pierwszy rzut oka może nie wydawać się to oczywiste, przygotowanie do

startu w konkretnych zawodach jest najefektywniejsze wtedy, gdy trening przeprowadzany jest w warunkach zbliżonych do tych, które można spotkać na trasie biegu. Z tego względu użytkownicy poszukujący trasy, zwykle mają co do niej pewne preferencje, które nie mogą zostać uwzględnione, ponieważ kryteria wyszukiwania tras w istniejących aplikacjach są bardzo ograniczone.

Po drugie, mimo że porównanie czasów osiągniętych na poszczególnych trasach jest dobrym sposobem na sprawdzenie swojej obecnej formy, istniejące aplikacje umożliwiają sprawdzenie rezultatu dopiero po zakończonym treningu, a nie w jego trakcie.

Z tych względów, głównym przedmiotem pracy jest zaprojektowanie i stworzenie aplikacji, której funkcjonalność jest udoskonalona i w efekcie nie posiada wymienionych problemów. Do każdej z tras udostępnianych przez społeczność zostaną przypisane pewne cechy. Ich określanie dokonywane będzie automatycznie, jednak biegacz tworzący trasę będzie miał możliwość skorygowania niedokładności we własnym zakresie. Są one następujące:

- całkowita długość wyrażona w kilometrach,
- nachylenie terenu,
- twardość nawierzchni wyrażona w procentach - określa jaka część całej trasy prowadzi przez grunt utwardzony.

Użytkownik ma możliwość określenia kryteriów wyszukiwania powiązanych z cechami trasy. Dodatkowo może on określić maksymalny promień wyszukiwania względem jego obecnej pozycji. W ten sposób otrzymuje on rezultat zawierający znajdujące się dostatecznie blisko trasy spełniające jego wymagania, przez co przygotowanie do zawodów może być efektywniejsze. Tempo biegacza może zmieniać się na różnych fragmentach trasy, a więc osoba zajmująca pierwszą pozycję w połowie zawodów, niekoniecznie je zwycięży. Mając to na uwadze możliwe jest udoskonalenie drugiego aspektu z aktualnie istniejących aplikacji. Na podstawie prób zawodników którzy ukończyli wcześniej konkretną trasę, użytkownik będzie informowany nie tylko o ostatecznie osiągniętej pozycji, lecz także o aktualnie zajmowanym miejscu w klasyfikacji na poszczególnych fragmentach trasy oraz o fakcie, że zyskał lub utracił pozycję. Taka symulacja sprawia wrażenie uczestnictwa w wirtualnych zawodach. Uczucie rywalizacji z innymi, może nie tylko urozmaicić trening, ale także sprawić, że za sprawą chęci wygranej, będzie on efektywniejszy.

1.2. Cele pracy

Do celów niniejszej pracy należą:

- **Opracowanie modelu danych pozwalającego na przechowywanie dodatkowych informacji o zapisywanych trasach wraz z rozwiązaniem problemu charakterystyki tras oraz wirtualnej rywalizacji** - pierwszym z działań które należy podjąć, jest zaprojektowanie modelu danych. Tworzone trasy muszą być zapisywane razem z cechami, które je opisują. Model powinien pozwalać na przeglądanie tras z uwzględnieniem wprowadzonych kryteriów wyszukiwania. Jako że aplikacja składa się z funkcjonalności, do których działania wymagana jest komunikacja pomiędzy użytkownikami, model danych musi zostać opracowany nie tylko dla aplikacji mobilnej, lecz także dla części serwerowej systemu. Następnie należy opracować logikę, która pozwoli przypisać trasie pewne cechy (zarówno automatyczne jak i manualne na podstawie opinii użytkownika). Charakterystyka będzie jedynie sugestią, dlatego użytkownik powinien mieć możliwość ich skorygowania. Ostatnim elementem jest zaproponowanie podejścia, które na podstawie poprzednich prób różnych użytkowników, pozwoli przeprowadzić swego rodzaju „wirtualny wyścig".
- **Stworzenie prototypu mobilnej aplikacji wspomagającej trening biegaczy** - Dotyczy implementacji założeń opracowanych w pierwszym celu oraz zbudowaniu systemu składającego się z aplikacji mobilnej przeznaczonej na system operacyjny Android i aplikacji serwerowej posiadającej dostęp do bazy danych. Całość powinna być zaprojektowana w sposób, który w przyszłości umożliwi ewentualną obsługę kolejnego mobilnego systemu operacyjnego bez modyfikacji istniejącego kodu źródłowego.
- **Ocena możliwości praktycznych stworzonego prototypu poprzez porównanie do istniejących aplikacji o podobnym zastosowaniu** - Na końcu stworzony system zostanie porównany z innymi, znajdującymi się już na rynku, rozwiązaniami w dziedzinie aplikacji treningowych. Na tej podstawie możliwa będzie ocena, czy rzeczywiście oferuje on rozszerzoną funkcjonalność.

1.3. Przegląd literatury

Niniejszy podrozdział zawiera pozycje, na które warto zwrócić uwagę, w przypadku potrzeby zgłębienia tematu podejmowanego w ramach pracy.

1.3.1. Wykorzystywanie nawigacji satelitarnej

GPS i inne satelitarne systemy nawigacyjne [1] - Tytuł poświęcony omówieniu zarówno podstaw jak i tajników działania systemów nawigacji satelitarnej. Autor udziela informacji o ich zasadach działania i budowie. Przedstawiono sposoby wyznaczania pozycji, położenia oraz prędkości obiektu, a także problemy które mogą pojawić się podczas wykonywania tych czynności.

1.3.2. Budowanie aplikacji mobilnych

1.3.3. Budowanie aplikacji serwerowych

1.3.4. Obsługa bazy danych

1.4. Układ pracy

TODO: poniższy tekst do aktualizacji Niniejszy rozdział jest wstępem do pracy. Określa on główną problematykę i zakres pracy, charakteryzuje jej cele oraz wymienia pozycje literackie, które szczegółowo opisują tematy tworzenia aplikacji oraz wykorzystania danych udostępnianych przez nawigację satelitarną. W rozdziale drugim opisany jest sposób, w jaki urządzenia mobilne używane są do wspomagania treningów biegowych. Zawiera on podstawowe informacje dotyczące zasady działania takich aplikacji, metod wykorzystania nawigacji satelitarnej, sposobów informowania użytkowników o rezultatach przeprowadzanych treningów. Omawia także słabe strony istniejących aplikacji, które posiadają mocną pozycję na rynku. W rozdziale trzecim skupiono się na procesie charakteryzowania tras. Omówiono wszystkie dane jakie może zawierać trasa, jakie cechy mogą zostać z nich wyodrębnione oraz sam sposób ich wyodrębnienia. Rozdział czwarty zaś, wyjaśnia jak zapisane dane, mogą być wykorzystane do przeprowadzenia symulacji rywalizacji pomiędzy zawodnikami. Omawia także potencjalne problemy, które można przy tym napotkać wraz z propozycją ich rozwiązania. W rozdziale piątym przedstawiono technologie użyte do stworzenia całego systemu: narzędzia programistyczne, język programowania użyty do stworzenia aplikacji mobilnej, aplikacji serwerowej, system baz danych, który przechowuje dane niezbędne do jej działania oraz bibliotekę umożliwiającą wyświetlanie map w graficznym interfejsie użytkownika. W rozdziale szóstym przedstawiono fazy budowy aplikacji: analizę wymagań,

jej architekturę, implementację oraz eksperyment testowy, który udowodni poprawność logiki odpowiedzialnej za wirtualną rywalizację. W ostatnim rozdziale zawarto podsumowanie. Wynika z niego, że udało się opracować model danych, który pozwolił przypisać do trasy pewne cechy ją charakteryzujące oraz przeprowadzić symulację rzeczywistej rywalizacji biegowej. Na podstawie porównania funkcji zaimplementowanych w ramach niniejszej z tymi, które zawierają istniejące aplikacje, wysnuto także wniosek mówiący, że udało się stworzyć aplikację służącą do treningów biegowych, która oferuje nowe oraz unikalne w swojej kategorii funkcjonalności.

2. Wykorzystanie urządzeń mobilnych i nawigacji satelitarnej do treningów biegowych

2.1. Nawigacja satelitarna

Nawigacja satelitarna jest systemem, który pozwala ustalić pozycję punktu w przestrzeni. Zasada jej działania opiera się na pomiarze odległości pomiędzy odbiornikiem reprezentującym punkt, którego pozycja ma zostać ustalona, a satelitami znajdującymi się na orbicie okołoziemskiej. Wyznaczając odległość pomiędzy tylko jedną satelitą a odbiornikiem, możliwe jest stwierdzenie, że poszukiwany punkt znajduje się gdzieś na powierzchni pewnej sfery o środku równym położeniu satelity w chwili wysłania sygnału oraz promieniowi równemu wyznaczonej odległości. Analogicznie, mając dostęp do dwóch satelitów, można otrzymać dwie sfery. Odbiornik znajduje się wtedy na okręgu powstałym z przecięcia obu sfer. Choć pozycja odbiornika jest uściślona względem pierwszego przypadku, to nadal nie można ustalić jej dokładnej pozycji. Skorzystanie z trzech satelitów. Skorzystanie z trzeciego satelity sprawia, że przecięcie wszystkich sfer wyznacza tylko dwa punkty, w których może znajdować się antena odbiornika. Jeden z tych punktów znajduje się bardzo wysoko nad ziemią, dlatego można go pominąć. W ten sposób możliwe ustalenie dokładnej pozycji odbiornika w dwóch wymiarach. Chcąc poznać także wysokość na której znajduje się punkt, konieczne jest skorzystanie z minimum czterech satelitów [1]. Pozycja użytkownika ustalona w dwóch wymiarach jest wystarczająca do przeprowadzenia procesu wirtualnej rywalizacji. Automatyczne przypisanie cechy, która reprezentuje poziom terenu przez który przebiega trasa wymaga jednak informacji o pozycji użytkownika w trzech wymiarach.

Do pewnego momentu wspieranie treningów za pomocą danych pochodzących z nawigacji satelitarnej wymagało zakupu przyrządu przeznaczonego konkretnie do tego celu. Intensywny rozwój technologiczny sprawił jednak, że moduł nawigacji satelitarnej zaczął być powszechnie stosowany w telefonach komórkowych, a więc jego posiadanie stało się niezwykle powszechne.

Warto zaznaczyć, że Globalny System Pozycyjny (GPS) jest niekiedy błędnie uogólniany do pojęcia nawigacji satelitarnej, gdy tak naprawdę jest on jedynie jednym z kilku istniejących systemów stosowanych obecnie w telefonach komórkowych:

- GPS - amerykański system nawigacji, który jako pierwszy został oddany do użytku. W chwili obecnej w jego skład wchodzi 30 satelitów, które mogą osiągnąć dokładność rzędu 5 metrów. Zdobył ogromną popularność i można stwierdzić, że jest on jednym z największych osiągnięć technicznych ubiegłego wieku [**?**],
- GLONASS - system nawigacji stworzony przez Rosję. Składa się z 24 satelitów, które zastosowaniom cywilnym pozwalają na osiągnięcie dokładności około pięciu metrów [1],
- BeiDou - chiński system nawigacyjny. Docelowo w jego obrębie ma działać 35 satelitów pozwalających osiągnąć dokładność około dziesięciu metrów [1, **?**],
- Galileo - europejski system nawigacji, który nadal jest w fazie budowy. Zakłada się, że jego precyzja będzie wynosić 4 metry. Zakończenie prac planowane jest na rok 2020 [**?**, 1].

Precyzja systemów nawigacji używanych obecnie w urządzeniach mobilnych jest nie mniejsza niż 10 metrów. Czynniki takie jak niekorzystne warunki atmosferyczne mogą jednak mieć negatywny wpływ na dokładność [1], co należy mieć na uwadze podczas projektowania aplikacji przeznaczonej treningom biegowym.

2.2. Działania na danych pochodzących z nawigacji satelitarnej

2.2.1. Obliczanie odległości pomiędzy dwoma punktami geograficznymi

Do obliczenia odległości pomiędzy dwoma punktami na sferze można skorzystać z **formuły haversine** [**?**]. Jej wzór prezentuje się następująco:

$$a = \sin^2(\frac{\Delta\phi}{2}) + cos\phi_1 \cdot cos\phi_2 \cdot sin^2(\frac{\Delta\lambda}{2}) \tag{2.1}$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \tag{2.2}$$

$$d = R \cdot c \tag{2.3}$$

gdzie ϕ jest szerokością geograficzną, λ długością geograficzną, R średnim promieniem Ziemi wynoszącym w zaokrągleniu 6371 kilometrów. Wzór zakłada, że miary wszystkich kątów podane są w radianach.

2.2.2. Wyznaczanie punktu oddalonego o podany dystans i kąt od punktu początkowego

Znając punkt początkowy, kierunek oraz odległość możliwe jest wyznaczenie punktu końcowego za pomocą następującego wzoru [**?**]:

$$\begin{cases} \varphi_2 = asin(sin\varphi_1 \cdot cos\delta + cos\varphi_1 \cdot sin\delta \cdot cos\theta) \\ \lambda_2 = \lambda_1 + atan2(sin\theta \cdot sin\delta \cdot cos\varphi_1, cos\delta - sin\varphi_1 \cdot sin\varphi_2) \end{cases} \tag{2.4}$$

gdzie ϕ jest szerokością geograficzną, λ długością geograficzną, θ określonym względem północy, zgodnie ze wskazówkami zegara kierunkiem wyrażonym w radianach, d odległością od punktu początkowego, R średnim promieniem Ziemi wynoszącym w zaokrągleniu 6371 kilometrów, δ odległością kątową ($\frac{d}{R}$). Wzór zakłada, że miary wszystkich kątów podane są w radianach.

2.3. Możliwe problemy i sposoby ich rozwiązania

Wykorzystywanie danych pochodzących z nawigacji satelitarnej do celu, w którym precyzja jest niezwykle istotna może przysporzyć pewnych problemów. W tym rozdziale opisano możliwe do napotkania problemy oraz propozycje ich rozwiązania.

2.3.1. Automatyczne określenie poziomu terenu w przypadku dostępu do jedynie trzech satelitów

Jak wspomniano w podrozdziale 2.1, mając jednoczesny dostęp do trzech satelitów, pomimo znajomości pozycji użytkownika na mapie, nie jest możliwe określenie jego wysokości nad poziomem morza. Z tego powodu nie można określić cechy trasy, która daje biegaczowi informacje na temat tego czy możliwe jest napotkanie na niej jakichś podbiegów lub zbiegów. Brakująca wysokość może zostać ustalona przy pomocy zewnętrznych serwisów udostępniających brakującą daną na podstawie szerokości oraz długości geograficznej. Przykładem narzędzia udostępniającego tę funkcjonalność jest serwis Google Maps [**?**]. Biorąc pod uwagę, że urządzenia mobilne są obecnie w stanie obsługiwać różne systemy nawigacji satelitarnej, których ilość dostępnych przekracza 20, prawdopodobieństwo posiadania jednoczesnego kontaktu z mniej niż czterema satelitami, a tym samym niemożności określenia wysokości użytkownika nad poziomem morza jest stosunkowo niskie. Z tego względu w aplikacji tworzonej w ramach niniejszej pracy nie zaimplementowano pobierania wysokości

z zewnętrznego serwisu. W przypadku gdy ilość jednocześnie dostępnych satelitów nie przekracza trzech, omawiana cecha trasy jest ustalana przez użytkownika.

2.3.2. Wahania pozycji przy braku ruchu

Pozycja wyznaczana przy użyciu nawigacji satelitarnej obarczona jest kilkumetrowym błędem. Oznacza to, że dokonując kilku odczytów, lokalizacja okaże się za każdym razem inna, pomimo faktu że w rzeczywistości odbiornik znajduje się za każdym razem w tym samym miejscu. Przykładowe odczyty dla nieporuszającego się odbiornika zostały przedstawione w tabeli 2.1, a ich wizualizacja na mapie umieszczono na rysunku 2.1. Odległość pomiędzy skrajnie oddalonymi od siebie punktami wynosi niespełna 10 metrów. Rozwiązaniem tego problemu jest branie pobranego punktu pod uwagę tylko wtedy, gdy jego odległość od punktu go poprzedzającego przekracza pewną wartość. Użytkownik nieporuszający się nie spełni tego warunku, co zapobiegnie wystąpieniu przedstawionego problemu.

Tabela 2.1: Współrzędne nieporuszającego się odbiornika.

L.p.	Szerokość geograficzna	Długość geograficzna
1.	51.95	19.204412
2.	51.949984	19.204416
3.	51.949975	19.204412
4.	51.949966	19.204412
5.	51.949942	19.204414
6.	51.949922	19.204446
7.	51.949921	19.204449
8.	51.949918	19.204456

Rysunek 2.1: Wizualizacja współrzędnych nieporuszającego się odbiornika na mapie.

Zignorowanie powyższego zjawiska, mogłoby doprowadzić do bardzo niekorzystnej sytuacji. Gdyby osoba tworząca trasę zatrzymała się w trakcie treningu, na przykład w celu odpoczynku lub zawiązania buta, zapisany przebieg treningu zawierałby skupisko fałszywych punktów. W celu wyeliminowania tego zagrożenia, podczas tworzenia trasy brane są pod uwagę tylko te punkty, które są oddalone od poprzedniego o więcej niż 10 metrów. Dzięki temu chwilowe zatrzymanie się podczas treningu, nie powoduje powstania przekłamań w generowanej trasie.

2.3.3. Zapobieganie oszustwom

Opieranie rywalizacji biegowej jednie na danych pobieranych z systemów nawigacji satelitarnej, sprawia że jej uczciwość może być zaburzona w stosunkowo łatwy sposób. Nic nie stoi na przeszkodzie, aby do pokonania trasy użyć chociażby roweru. Możliwe jest także manualne wprowadzanie danych lokalizacyjnych do urządzenia [**?**], co pozwoli uzyskać dowolnie dobry wynik bez podejmowania jakiejkolwiek aktywności fizycznej. Zagrożenia tego można uniknąć na dwa sposoby:

- **Porównywanie osiąganych przez użytkowników rezultatów z wynikami najlepszych na świecie biegaczy** - Zapisanie w aplikacji zbioru światowych rekordów biegowych na konkretnych dystansach i porównywanie ich do wyników uzyskiwanych przez użytkowników jest pierwszym ze sposobów na zapobieganie oszustwom. Rezultaty znacząco lepsze od tych, które zostały osiągnięte na atestowanych zawodach z całą pewnością mogą zostać uznane jako osiągnięte nieuczciwie. Choć rozwiązanie to zapobiegnie zapisywaniu w rankingu czasów trasy, które są wręcz niemożliwe do osiągnięcia, nie sprawdzi się w przypadkach gdy sfałszowany wynik mieści się w granicach zdrowego rozsądku.
- **Skorzystanie z czujników dostępnych w urządzeniu** - Akcelerometr oraz żyroskop są jednymi z wielu czujników umieszczanych w produkowanych obecnie telefonach komórkowych. Ich zadaniem jest mierzenie przyspieszenia liniowego oraz położenia kątowego. Umożliwiają zatem pozyskanie informacji o ruchach jakim poddane jest urządzenie w trakcie trwania treningu. Opracowanie odpowiedniego algorytmu oraz skorzystanie z sieci neuronowych pozwoliłyby stwierdzić czy dane pobrane z czujników w czasie odbywania treningu, zgadzają się z tymi, które generowane są zwykle w czasie biegu i na tej podstawie dokonać akceptacji lub unieważnienia próby użytkownika. W przeciwieństwie do poprzedniego podejścia, bieżące rozwiązanie okazałoby się skuteczne także w przypadkach, w których dane lokalizacyjne zostały wprowadzone do urządzenia manualnie oraz gdy osiągnięty na trasie rezultat nie jest lepszy niż ogólnoświatowe rekordy. Rozwiązanie tego problemu obejmuje zagadnienia z dziedziny sztucznej inteligencji i wykracza poza zakres niniejszej pracy, dlatego też nie zostało zaimplementowane w tworzonej aplikacji.

2.4. Istniejące rozwiązania w dziedzinie aplikacji treningowych

Na rynku aplikacji znajduje się wiele aplikacji, których przeznaczeniem jest wspieranie treningów biegowych. Pozwalają one użytkownikom na dogłębną analizę aktywności sportowych oraz śledzenie treningów zarówno swoich jak i udostępnianych przez innych biegaczy. Funkcjonalności te są bardzo powszechne. W chwili pisania niniejszej pracy, nie udało się znaleźć aplikacji treningowej, która w pełni oferuje funkcje zawarte w nazwa-mojej-aplikacji, dlatego do niniejszego przeglądu wybrano trzy przykłady aplikacji, których zakres działania jest jak najbardziej do niej zbliżony, a jednocześnie cieszących się wysoką oceną oraz popularnością w sklepie z aplikacjami Google Play [**?**].

2.4.1. Endomondo

Jedną z najwcześniej wydanych (pierwsze wersja została udostępniona w listopadzie 2007 roku), a zarówno najbardziej popularnych na świecie aplikacji jest Endomondo [**?**]. Pozwala na śledzenie aktywności w ponad 60 dyscyplinach sportowych, włączając w to bieganie. W aplikacji rozwinięto aspekt społecznościowy - znajomi mają możliwość wzajemnego przeglądania historii swoich treningów. Ich statystyki mogą być także udostępniane na mediach społecznościowych. Ponadto istnieje możliwość tworzenia własnych tras, które inni biegacze mogą następnie przeglądać w formie listy i używać do własnych treningów. Niestety możliwe jest zobaczenie tylko całego zbioru tras, znajdujących się w pewnej nieznanej odległości od użytkownika. Stanowi to problem, ponieważ nie mogąc określić dokładnie obszaru wyszukiwania, biegacze nie mogą być pewni czy zobaczyli już wszystkie trasy, które mogłyby ich potencjalnie zainteresować czy może istnieje ich więcej, ale znajdują się poza nieznanym obszarem wyszukiwania. Producent nie udostępnił także możliwości filtrowania wyników wyszukiwania ze względu na długość trasy, rodzaj nawierzchni czy poziom terenu. Podczas tworzenia trasy zapisywana jest co prawda jej całkowita długość, jednak służy ona tylko poinformowaniu użytkownika, a nie wyszukiwaniu. Aplikacja nie pozwala także na jakiekolwiek porównywanie wyników, które biegacze osiągają na trasie.

2.4.2. Strava

Strava [**?**] jest aplikacją, która w momencie pisania niniejszej pracy wspiera treningi w 24 dyscyplinach sportowych. Oprócz podstawowej funkcjonalności aplikacji treningowych, oferuje także analizę treningów na podstawie rozmaitych statystyk i wykresów. Aplikacja

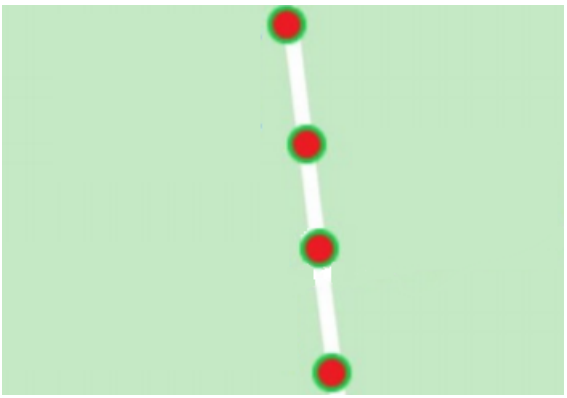
umożliwia także funkcję tworzenia i zapisywania tras. Oprócz samego jej przebiegu, zapisywana jest także jej długość oraz nachylenie terenu. Niestety tak jak w przypadku aplikacji Endomondo, producent nie umożliwił filtrowania tras według żadnej z tych cech. Podczas przeglądania tras, ukazują się one na interaktywnej mapie, którą można przesuwać. Nie występuje tutaj zatem znany z Endomondo problem niemożności określenia obszaru w którym wyszukiwane są trasy. Rozszerzenie obszaru poszukiwania ogranicza się do przesunięcia mapy na ekranie urządzenia. Zapisywanie czasów użytkowników którzy ukończyli trasę na serwerze, pozwala użytkownikom na wyświetlenie rankingu trasy i porównanie swojej dyspozycji do innych biegaczy. Funkcjonalność ta daje poczucie pewnego rodzaju rywalizacji, jednak sprawdzenie uzyskanego wyniku możliwe jest dopiero po ukończeniu treningu, użytkownik nie otrzymuje na ten temat informacji w czasie jego trwania.

2.4.3. MapMyRun

Przeznaczeniem aplikacji MapMyRun [**?**] jest przede wszystkim śledzenie aktywności związanych ze spacerowaniem i bieganiem. W tym przykładzie użytkownicy także mają możliwość przeglądania i analizowania swoich poprzednich treningów. Usprawnieniem jest fakt, iż trasy można filtrować na podstawie ich długości. Zabrakło niestety filtra poziomu terenu, mimo że informacja ta jest zapisywana podczas tworzenia trasy. Podczas wyszukiwania tras pojawia się problem znany z aplikacji Endomondo - prezentowane trasy należą do obszaru, którego granice oddalone są od użytkownika o nieznaną odległość, przez co biegacze nie mogą być pewni czy zostały im wyświetlone wszystkie godne uwagi trasy czy może część z nich znajduje się poza obszarem poszukiwań. Element rywalizacji jest zorganizowany podobnie jak w aplikacji Strava - wyniki osiągane przez użytkowników są zapisywane na serwerze, dzięki czemu mają oni możliwość sprawdzenia swojej dyspozycji po zakończonym treningu.

3. Tworzenie i charakterystyka tras

Tworzenie trasy polega na zapisaniu w systemie pozycji, ustalonych przy użyciu nawigacji satelitarnej, na których użytkownik znajdował się w trakcie treningu tak jak zostało to pokazane na rysunku przedstawiającym fragment utworzonej trasy 3.1, na którym linią ciągłą zaznaczono rzeczywistą trasę pokonaną przez biegacza, a kropkami reprezentację trasy zapisaną w systemie. Zapamiętywane są tylko te punkty, które znajdują się w odległości nie mniejszej niż 10 metrów od swojego poprzednika. Zaproponowana minimalna odległość pomiędzy punktami, pozwoli na osiągnięcie dostatecznie dokładnego przebiegu trasy, przy jednoczesnym zapobiegnięciu wystąpienia problemu, opisanego w rozdziale 2.3.2.



Rysunek 3.1: Fragment utworzonej trasy. [Opracowanie własne]

Zawody biegowe w których biegacze biorą udział, różnią się trasami, na jakich są przeprowadzane. Ich dystans może wynosić od kilku do kilkudziesięciu kilometrów. Zawody odbywające się w mieście zwykle odbywają się na płaskiej nawierzchni asfaltowej, podczas gdy biegi organizowane w górach wymagają często od zawodników podbiegania pod jakiś szczyt górski lub zbiegania z niego. Nie inaczej jest z przygotowaniem do konkretnych zawodów. Chcąc jak najefektywniej przygotować się do startu, biegacz powinien trenować w warunkach zbliżonych do tych, które może spotkać na zawodach. Z tego względu aplikacja tworzona w ramach niniejszej pracy umożliwia użytkownikom wyszukiwanie tras treningowych, które spełnią ich preferencje. Niniejszy rozdział jest realizacją części pierwszego celu przedstawionego w rozdziale 1.2. Omówiono w nim cechy, które mogą zostać

przypisane do trasy, sposób ich wyznaczenia na podstawie danych lokalizacyjnych oraz kryteria wyszukiwania używane podczas przeglądania tras.

3.1. Możliwe cechy trasy

Do każdej trasy mogą zostać przypisane 3 cechy.

- **Dystans** - określa jak długa jest trasa. Wartość wyrażona jest w kilometrach.
- **Poziom terenu** - Reprezentuje zmianę wysokości nad poziomem morza pomiędzy początkiem a końcem trasy. Wartości które może przyjąć ta cecha to:
  - Wyrównany - gdy wysokość nad poziomem morze nie zmienia się lub zmienia się nieznacznie,
  - Rosnący - gdy wysokość nad poziomem morza wzrasta,
  - Malejący - gdy wysokość nad poziomem morza maleje.
- **Twardość nawierzchni** - Wyraża jaka część całej trasy przebiega przez nawierzchnię utwardzoną (na przykład asfalt lub kostka brukowa) w stosunku do nawierzchni nieutwardzonej (na przykład piasek lub trawa). Wartość wyrażona jest w procentach. Przykładowo wartość 60% dla trasy o dystansie 10 kilometrów oznacza, że 6 kilometrów prowadzi przez nawierzchnię utwardzoną, a 4 kilometry przez nieutwardzoną.

3.2. Przypisanie cech do trasy

3.2.1. Długość trasy

Długość trasy przypisywana jest bezpośrednio po ukończeniu treningu. Jej wyznaczenie polega na zsumowaniu odległości pomiędzy wszystkimi następującymi po sobie punktami przy użyciu wzorów 2.1, 2.2 i 2.3.

3.2.2. Poziom terenu

Określenie poziomu terenu odbywa się poprzez obliczenie różnicy wysokości nad poziomem morza pomiędzy punktem rozpoczynającym oraz kończącym trasę i przypisanie odpowiednio poziomu:

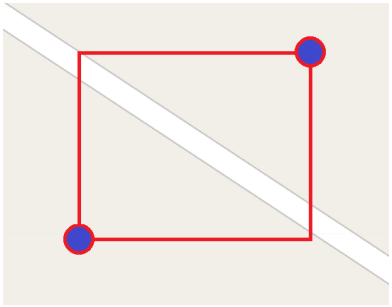
- **wyrównanego** - gdy różnica nie przekracza 10%,

- **rosnącego** - gdy wysokość punktu końcowego jest większa od wysokości punktu początkowego o więcej niż 10%,
- **malejącego** - gdy wysokość punktu początkowego jest większa od wysokości punktu końcowego o więcej niż 10%.

Należy mieć na uwadze, że w przypadku wystąpienia problemu opisanego w rozdziale 2.3.1, cecha ta będzie musiała zostać ustalona przez użytkownika.

3.2.3. Twardość nawierzchni

W celu umożliwienia automatycznego przypisania twardość nawierzchni, posłużono się serwisem OpenStreetMap [**?**]. Narzędzie to pozwala scharakteryzować podłoże ustalonego przez klienta obszaru. Musi on być zdefiniowany w formie prostokąta, a odbywa się to poprzez dostarczenie serwisowi współrzędnych geograficznych jego lewej dolnej oraz prawej górnej krawędzi. Zdefiniowane przykładowego obszaru zostało pokazane na rysunku 3.2. Kropkami oznaczono podane współrzędne geograficzne, natomiast linia ciągła definiuje obszar, który zostanie poddany analizie. W odpowiedzi klient otrzymuje charakterystykę wszystkich dróg, które znalazły się w zdefiniowanym obszarze [**?**]. Rodzaj nawierzchni nie jest zwracany w formie „utwardzona" bądź „nieutwardzona", lecz w formie sprecyzowanej, na przykład „asfalt" lub „trawa", dlatego pierwszym krokiem którego trzeba się podjąć po otrzymaniu odpowiedzi, jest przyporządkowanie etykiety „utwardzona" bądź „nieutwardzona" dla każdego z otrzymanych obiektów. Przyporządkowanie wszystkich możliwych do otrzymania rodzajów nawierzchni zostało pokazane w tabeli 3.1 [**?**].



Rysunek 3.2: Obszar poddany analizie [Opracowanie własne]

Tworzone przez użytkowników trasy zapisywane są w formie punktów, dlatego przed skorzystaniem z serwisu OpenStreetMap, należy zdefiniować wspomniany prostokątny obszar poszukiwań. Korzystając ze wzoru [**?**] wyznaczane są więc 2 nowe punkty: oddalony o 10

Tabela 3.1: Przyporządkowanie konkretnego rodzaju nawierzchni do jej reprezentacji w tworzonym systemie [**?**]

Nawierzchnia utwardzona	Nawierzchnia nieutwardzona
paved, asphalt, concrete, concrete:lanes, concrete:plates, paving_stones, sett, unhewn_cobblestone, cobblestone, metal, wood, metal_grid	unpaved, compacted, fine_gravel, gravel, pebblestone, dirt, earth, grass, grass_paver, ground, mud, sand, woodchips, snow, ice, salt, clay, tartan, artifical_turf, decoturf, carpet

.

metrów i 225 stopni względem północy oraz oddalony o 10 metrów i 45 stopni względem północy. Po ustaleniu rodzaju nawierzchni dla każdego z punktów trasy, możliwe jest wyliczenie wartości procentowej, a tym samym określenie cechy według wzoru:

$$Twardosc\ nawierzchni = \frac{u}{u+n} \cdot 100, \tag{3.1}$$

gdzie u oznacza liczbę obiektów z etykietą „utwardzona", n liczbę obiektów z etykietą „nieutwardzona". Z uwagi na czytelność wynik zaokrąglany jest do liczby całkowitej. Należy mieć na uwadze, że OpenStreetMap jest projektem tworzonym przez społeczność i dla pewnych dróg rodzaj nawierzchni mógł być przypisany błędnie bądź nie przypisany wcale. Wartość określonej cechy powinna być zatem traktowana w formie propozycji, którą użytkownik tworzący trasę może poprawić wedle własnego uznania.

3.3. Kryteria wyszukiwania tras

Kryteria wyszukiwania są ściśle powiązane z cechami opisanymi w rozdziale 3.1. Dokonano jedynie kilku usprawnień mających na celu wygodę użytkownika.

- Kryterium **długości trasy** przyjmuje wartości „od" oraz „do". Oznacza to, że lista wynikowa zawiera trasy nie krótsze niż pierwsza z nich, a jednocześnie nie dłuższe niż druga z nich. Największa wartość, którą można określić wynosi 20 kilometrów. Po jej przekroczeniu, przy wyszukiwaniu pod uwagę brane są trasy o nieograniczonej maksymalnej długości,
- Przy filtrowaniu ze względu na **poziom terenu** oprócz konkretnych wartości cechy możliwe jest także wybranie opcji „każdy". Oznacza to, że trasy nie będą filtrowane według tej cechy.
- **Twardość nawierzchni** podobnie jak w przypadku długości trasy określana jest jako zakres „od, do". Wartość maksymalna wynosi 100%.

Oprócz tego użytkownik musi określić maksymalny promień wyszukiwania względem jego obecnej lokalizacji. Dzięki temu jest on świadomy, że zostały mu pokazane wszystkie trasy z interesującego go obszaru. Nie bez znaczenia jest także fakt, że zapobiegnie to pobieraniu z bazy danych wszystkich istniejących tras, co miałoby znaczący wpływ na wykorzystanie łącza oraz wydajność zarówno aplikacji serwerowej jak i aplikacji mobilnej.

4. Symulacja rywalizacji na podstawie zgromadzonych danych

Funkcja wirtualnej rywalizacji daje użytkownikom uczucie uczestniczenia w prawdziwych zawodach. Współzawodnictwo z innymi biegaczami może sprawić, że trening będzie nie tylko efektywniejszy, ale także przyjemniejszy. Jest też rozwiązaniem dla osób, które chcą sprawdzić swoją dyspozycję bez ponoszenia kosztów podróży oraz wpisowego na prawdziwe zawody. Ponadto ze względu, że w wirtualnym wyścigu „bierze udział" także najlepszy poprzedni czas użytkownika na danej trasie, ma on możliwość zweryfikowania czy jego forma staje się coraz lepsza. Niniejszy rozdział jest kontynuacją realizacji pierwszego celu przedstawionego w rozdziale 1.2. Omówiono w nim sposób w jaki przeprowadzana jest wirtualna rywalizacja oraz opisano w jaki sposób użytkownik informowany jest o zachodzących w trakcie wyścigu zdarzeniach.

4.1. Sposób przeprowadzenia procesu wirtualnej rywalizacji

W momencie gdy użytkownik wybierze jedną z udostępnionych przez społeczność tras i rozpocznie trening, punkty których przeznaczeniem było do tej pory jedynie reprezentowanie przebiegu trasy, przyjmują rolę punktów kontrolnych. Zadaniem biegacza chcącego wziąć udział w wirtualnej rywalizacji jest przebiegnięcie obok wszystkich punktów w kolejności, w której zostały one stworzone. Przebieg takiego treningu musi więc pokrywać się z przebiegiem treningu użytkownika, który stworzył trasę. Aby następny w kolejności punkt kontrolny został oznaczony jako „osiągnięty", odległość pomiędzy aktualną pozycją użytkownika, a pozycją punktu kontrolnego obliczona za pomocą wzorów 2.1, 2.2 i 2.3 musi być nie większa niż 10 metrów.

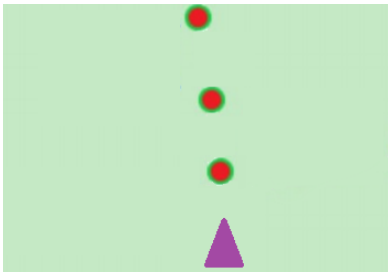
Za każdym razem gdy punkt kontrolny zostaje „osiągnięty", zapisywany jest aktualny czas użytkownika. Zapisywanie osiągniętego czasu ma miejsce również podczas tworzenia trasy w

momencie zapisywania pozycji punktów. Posiadając dane o czasach użytkowników na każdym z punktów kontrolnych trasy, możliwe jest przeprowadzenie procesu wirtualnej rywalizacji.

Wraz z rozpoczęciem treningu użytkownik „osiąga" pierwszy punkt kontrolny (będący jednocześnie pierwszym punktem zapisanej trasy). W tym momencie zostaje on dodany na koniec rankingu. Sytuacja ta została przedstawiona w tabeli 4.1 oraz na rysunku 4.1. Trójkątny kształt oznacza pozycję użytkownika, natomiast kropki reprezentują punkty kontrolne. Za każdym razem gdy biegacz „osiąga" następny w kolejności punkt kontrolny, ranking aktualizowany jest tak, aby wskazywać aktualnie przez niego zajmowaną pozycję. Oznacza to, że miejsce w rankingu może (choć nie musi) zmieniać się za każdym razem, gdy „osiągnięty" jest punkt kontrolny. Sytuacja zmiany pozycji została przedstawiona w tabeli 4.2 oraz na rysunku 4.2. Ostatni punkt kontrolny jest jednocześnie końcowym punktem trasy, a więc „osiągnięcie" go doprowadza do wyznaczenia końcowego rankingu trasy.

Tabela 4.1: Ranking na początku rywalizacji biegowej.

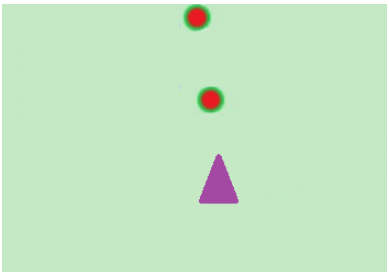
Zajmowana pozycja	Użytkownik
1	Użytkownik A
2	Użytkownik B
3	Użytkownik C
4	Bieżący użytkownik



Rysunek 4.1: Początek rywalizacji biegowej.

Tabela 4.2: Ranking po „osiągnięciu" kolejnego punktu kontrolnego.

Zajmowana pozycja	Użytkownik
1	Użytkownik B
2	Użytkownik A
3	Bieżący użytkownik
4	Użytkownik C



Rysunek 4.2: Rywalizacja biegowa po osiągnięciu kolejnego punktu.

Tak jak zostało wspomniane we wstępie do niniejszego rozdziału, użytkownik ma możliwość rywalizacji „ze samym sobą", a więc oprócz próby obecnej w rankingu może znajdować się także próba biegacza dodana w czasie treningu odbytego wcześniej. Takie zjawisko może mieć jednak miejsce tylko w czasie trwania treningu. Oznacza to, że po jego zakończenia musi zajść jeden z następujących scenariuszy:

- użytkownik nie odbył wcześniej treningu w ramach trasy - bieżąca próba jest zapisywana w systemie,
- użytkownik odbył wcześniej trening w ramach trasy, jego końcowy rezultat był lepszy niż w próbie bieżącej - bieżąca próba zostaje odrzucona,
- użytkownik odbił wcześniej trening w ramach trasy, jego końcowy rezultat był gorszy niż w próbie bieżącej - bieżąca próba zastępuje w systemie próbę poprzednią.

4.2. Informowanie użytkownika o rezultatach

Podstawowym sposobem informowania biegacza jest wyświetlanie stanu treningu na ekranie urządzenia. Użytkownik ma do dyspozycji interaktywną mapę na której pokazana jest jego aktualna pozycja oraz wszystkie punkty kontrolne. Za każdym razem gdy następny w kolejności punkt kontrolny zostanie „osiągnięty", zostaje on usunięty z mapy, aby użytkownik był świadom, że może udać się w kierunku kolejnego. Ponadto na ekranie wyświetlony jest ranking bieżącej rywalizacji biegowej. Przedstawia on czas osiągnięty na ostatnim punkcie kontrolnym oraz nazwy użytkownika wszystkich biegaczy, których próba zapisana jest w systemie. Miejsca w rankingu aktualizowane są za każdym razem gdy biegacz, który odbywa trening, osiągnie następny punkt kontrolny.

Biegacze często korzystają w czasie treningu z zestawu słuchawkowego. Wobec tego w celu zwiększenia immersji płynącej z wirtualnej rywalizacji, użytkownik informowany jest o rezultatach poprzez odtworzenie głosowych komunikatów. Za każdym razem gdy użytkownik „osiągnie" punkt kontrolny odtwarzana jest informacja:

- **o aktualnie zajmowanej pozycji**, gdy pozycja zajmowana na właśnie „osiągniętym" punkcie kontrolnym jest taka sama jak na poprzednim punkcie kontrolnym,
- **o liczbie zdobytych pozycji**, gdy pozycja zajmowana na właśnie „osiągniętym" punkcie kontrolnym jest lepsza niż na poprzednim punkcie kontrolnym,
- **o liczbie utraconych pozycji**, gdy pozycja zajmowana na właśnie „osiągniętym" punkcie kontrolnym jest słabsza niż na poprzednim punkcie kontrolnym.

5. Zastosowane technologie

W niniejszym rozdziale opisane zostały technologie użyte podczas tworzenia zarówno aplikacji mobilnej jak i aplikacji webowej.

5.1. Oprogramowanie

Visual Studio Community 2019?? - Zintegrowane środowisko programistyczne wydane przez firmę Microsoft [**?**]. Choć pozwala na pracę z wieloma językami programowania, jego przeznaczeniem jest głównie tworzenie oprogramowania w języku C# [**?**]. Visual Studio pozwala także na generowanie kodu, wykonywanie operacji bezpośrednio na serwerze bazodanowym, pracę z systemami kontroli wersji oraz korzystanie z wielu udogodnień narzędzi wspomagających pracę programisty. Istnieje także możliwość dodatkowego rozszerzenia funkcjonalności środowiska programistycznego za pomocą oddzielnych rozszerzeń. Dzięki wsparciu dla wielu technologii, możliwe jest tworzenie oprogramowania o rozmaitym przeznaczeniu, zaczynając od programów przeznaczonych na komputery i urządzenia mobilne, przez aplikacje webowe, aż po gry wideo i rozwiązania wykorzystujące nauczanie maszynowe.

Tworzony w Visual Studio program podzielony jest na **projekty** (ang. *projects*). Każdy projekt reprezentuje niezależny, oddzielnie kompilowany kod źródłowy. Zbiór projektów, które ze sobą współpracują są zwykle grupowane w jedno **rozwiązanie** (ang. *solution*). Pozwala ono programistom w łatwy sposób dodawać lub usuwać projekty a także zarządzać procesem ich kompilacji.

5.2. Aplikacja mobilna

Do stworzenia części mobilnej budowanego systemu wybrano technologię **Xamarin** [**?**], wydaną przez firmę Microsoft [**?**]. Wykorzystuje ona kod pisany w języku C# [**?**] do implementacji logiki aplikacji oraz kod XML **??** do tworzenia interfejsów użytkownika. Zdecydowano się na ten wybór przede wszystkim z powodu możliwości budowy aplikacji

wieloplatformowych, jakie udostępnia ta technologia. Jest to szczególnie ważne, ponieważ pomimo że w obecnej fazie aplikacja jest przeznaczona tylko na urządzenia mobilne z systemem operacyjnym Android **??**, zakłada się, że w przyszłości może ona wspierać także system operacyjny iOS **??** oraz inteligentne zegarki.

Istnieją dwa podejścia budowania aplikacji wieloplatformowych w Xamarin:

- Xamarin Forms - Pozwala współdzielić kod źródłowy odpowiadający zarówno za logikę aplikacji jak i interfejs użytkownika. Podczas tworzenia interfejsu możliwe jest używanie tylko pewnego zbioru kontrolek, które korzystają z natywnych funkcjonalności wspólnych dla wszystkich obsługiwanych w Xamarin systemów operacyjnych.
- Xamarin Native - Pozwala współdzielić kod źródłowy odpowiadający za logikę. interfejs użytkownika tworzony jest dla każdego systemu operacyjnego oddzielnie. Dzięki temu w aplikacji mogą być wykorzystywane wszystkie funkcje udostępniane przez system operacyjny.

W celu uniknięcia problemów z obsługą nawigacji satelitarnej, wyświetlaniem rankingu oraz punktów kontrolnych na mapie w tworzonej użyto podejścia Native, a więc kontrolki używane w widokach aplikacji, są kontrolkami należącymi bezpośrednio do systemu Android.

5.3. Aplikacja webowa

Do stworzenia części serwerowej budowanego systemu wybrano język C# [**?**] oraz szkielet aplikacyjny (ang. *framework*) **ASP.NET Core** [**?**] służący do budowy aplikacji internetu rzeczy, aplikacji internetowych, programów działających w chmurze obliczeniowej czy (co istotne w tym przypadku) części serwerowych aplikacji mobilnych. ASP.NET Core został wydany w 2016 roku jako nowa i udoskonalona odsłona szkieletu aplikacyjnego ASP.NET. Nowa wersja zapewnia wieloplatformowość (aplikacje mogą być tworzone i uruchamiane w systemach Windows, Linux i macOS). Ważną cechą tej technologii jest nastawienie na wydajność - w przeciwieństwie do swojego poprzednika ASP.NET Core nie jest oparty na jednej bibliotece, ale na zestawie pakietów zawierających różnorodne funkcjonalności. Dzięki temu do aplikacji dodane są tylko te zależności, które rzeczywiście są używane, a w efekcie jest ona bardziej wydajna.

Komunikacja pomiędzy aplikacją serwerową a bazą danych odbywa się poprzez bibliotekę **Entity Framework**, która jest platformą mapowania obiektowo-relacyjnego. Pozwala ona

zastąpić konieczność pisania kodu SQL (*Structured Query Language*) wykonywaniem operacji bazodanowych poprzez modyfikowanie stanu obiektów w programie.

5.4. System zarządzania bazą danych

Jako oprogramowanie odpowiedzialne za zapisywanie, odczytywanie oraz zarządzanie danymi wybrano **SQL Server ??**. Jest to system zarządzania relacyjną bazą danych wydany przez firmę Microsoft. SQL Server jest jednym z najczęściej stosowanych rozwiązań w tej dziedzinie. Swoją pozycję zawdzięcza sobie wysoką wydajnością, ścisłą integracją z pozostałymi produktami Microsoft (na przykład chmurą Azure), łatwością w instalacji i eksploatacji (istnieją dedykowane narzędzia, które znacznie upraszczają zarządzania bazami danych).

5.5. Obsługa map

W systemie posłużono się dwoma różnymi rozwiązaniami oferującymi obsługę i dostęp do mapy świata.

5.5.1. Wyświetlanie mapy w aplikacji mobilnej

Mapa wyświetlana jest na ekranie urządzenia mobilnego dzięki wykorzystaniu biblioteki serwisu **Google** Maps [**?**]. Charakteryzuje się ona wysokim poziomem interaktywności. Użytkownik ma możliwość przesuwania, oddalania, przybliżania mapy oraz śledzenia własnej pozycji. Programista ma także możliwość wyświetlania na mapie rozmaitych dodatkowych grafik takich jak: ikony, linie, figury geometryczne, markery i personalizowania jej według własnych potrzeb.

W chwili pisania niniejszej pracy dostęp do wymienionych funkcjonalności jest całkowicie darmowy. Dodatkowej opłaty wymagają funkcje związane z wskazywaniem drogi do konkretnego punktu oraz pokazywaniem szczegółowych informacji o konkretnych lokalizacjach.

5.5.2. Automatyczne ustalanie twardości nawierzchni

OpenStreetMap [**?**] jest projektem mającym na celu stworzenie darmowej mapy całego globu. Mapa jest tworzona przez społeczność. Oznacza to, że każdy ma możliwość

modyfikowania jej i wprowadzania poprawek. Dane mogą być wprowadzane na podstawie odczytów z systemów nawigacji satelitarnej, innych map lub na podstawie własnych doświadczeń. Istotną cechą mapy jest jej szczegółowość. W jej tworzeniu bierze udział wiele osób i to od nich samych zależy jakie informacje zostaną umieszczone na mapie. W efekcie oprócz dróg i budynków można na niej znaleźć nawet takie obiekty jak skrzynki pocztowe, żywopłoty czy bocianie gniazda. [**?**].

OpenStreetMap udostępnia także programistom możliwość pozyskania danych poprzez kierowanie do serwisu zapytań HTTP. Funkcjonalność ta została użyta w aplikacji tworzonej w ramach niniejszej pracy do określania twardości terenu na tworzonej trasie. Aspekt ten opisano w rozdziale 3.2. Oprócz charakterystyki rodzaju terenu w odpowiedziach otrzymywanych z serwisu można znaleźć także wszystkie inne informacje, które użytkownicy uwzględnili tworząc konkretny fragment mapy. Przyjmuje się, że bezpieczna ilość zapytań, którą można kierować do serwisu bez jego obciążenia wynosi 10000 [**?**].

6. Dokumentaja techniczna

Niniejszy rozdział stanowi objaśnienie, w jaki sposób system został stworzony. Zawiera opis wymagań oraz szczegóły implementacyjne dotyczące budowy poszczególnych warstw przy wykorzystaniu technologii opisanych w rozdziale 5.

6.1. Projekt systemu

6.1.1. Opis założeń

moja-aplikacja składa się z aplikacji przeznaczonej na urządzenia z systemem Android [**?**] oraz wspierającej ją aplikacji webowej. Komunikacja pomiędzy nimi odbywa się poprzez protokół HTTP (*Hypertext Transfer Protocol*), a sam interfejs aplikacji serwerowej został wykonany w stylu **REST** (*Representational state transfer*). Oznacza to między innymi, że konkretny zasób na serwerze jest identyfikowany na podstawie przypisanego mu URI (*Uniform Resource Identifier*), a użytkownik odpytujący serwer jest identyfikowany na podstawie parametru zawartego w nagłówkach żądania. Wymiana danych pomiędzy obiema aplikacjami odbywa się przy pomocy formatu JSON (*JavaScript Object Notation*). [**?**]

Aplikacja mobilna powinna:

- Wspierać wieloplatformość - zastosowana architektura powinna oddzielać logikę biznesową aplikacji od interfejsu graficznego. W efekcie dzięki zastosowaniu technologii Xamarin, przeniesienie aplikacji na inny mobilny system operacyjny powinno ograniczać się do zbudowania jedynie widoku przeznaczonego pod ten system.
- Pozwalać użytkownikom tworzyć trasy, a także umożliwiać manualne lub automatyczne określenie ich cech.
- Pozwalać przeglądać trasy na podstawie zdefiniowanych przez użytkownika kryteriów wyszukiwania.
- Pozwalać na przeprowadzenie treningu na wybranej trasie wraz z elementem wirtualnej rywalizacji.

- Informować użytkownika o rezultatach zarówno poprzez informacje wyświetlane na ekranie jak i odtwarzane głosowo.
- Reagować na błędy - na przykład nieudane połączenie z aplikacją serwerową.

Aplikacja serwerowa powinna:

- Pozwolić wyszukiwać trasy na podstawie otrzymanych kryteriów
- Zapisywać trasy stworzone przez użytkowników
- Zapisywać próby użytkowników na trasach

6.1.2. Podział projektu

Dzięki skorzystaniu z technologii umożliwiającej tworzenie aplikacji mobilnych, aplikacji serwerowych oraz zintegrowanego środowiska programistycznego pochodzącego od jednego producenta, możliwe było tworzenie obu aplikacji w ramach pojedynczego rozwiązania (ang. *solution*). Pozwoliło to na pisanie kodu źródłowego, kompilowanie oraz uruchamianie w trybie debugowania obu aplikacji w jednym momencie. Nie było więc potrzeby korzystania z dwóch różnych środowisk programistycznych jednocześnie, co okazało się bardzo znaczącym udogodnieniem w kontekście wydajnościowym.

Stworzona solucja zawiera następujące projekty:

- Api - zawiera część serwerową systemu.
- Core - zawiera logikę biznesową oraz model danych części aplikacji mobilnej systemu.
- MobileAndroid - zawiera kod interfejs użytkownika aplikacji przeznaczonej na system operacyjny Android.

Rozdzielenie logiki biznesowej oraz modelu danych od widoku aplikacji mobilnej sprawia, że dodanie wsparcia dla dodatkowych mobilnych systemów operacyjnych ogranicza się do stworzenia w solucji nowego projektu zawierającego jedynie interfejs użytkownika oraz wykorzystanie projektu Core do jego obsługi.

6.1.3. Przygotowanie aplikacji do działania

Aplikacja serwerowa

Przed zbudowaniem aplikacji serwerowej należy ustawić w pliku konfiguracyjnym projektu (*appsettings.json*) parametry połączenia (ang *connection string*) bazy danych. Przykład konfiguracji przedstawiono na listingu 6.1.

```
{
  "ConnectionStrings": {
    "Default": "data source=.SQLEXPRESS; initial catalog=
    BscThesisDb; integrated security=SSPI"
  }
}
```

Listing 6.1: Plik konfiguracyjny aplikacji serwerowej

Następnie należy wykonać polecenie *dotnet build*. Spowoduje ono pobranie potrzebnych zależności oraz zbudowanie projektu. Ostatnim krokiem jest wywołanie polecenia *dotnet ef database update* w celu stworzenia bazy danych i potrzebnych tabel.

Aplikacja mobilna

Przed zbudowaniem aplikacji mobilnej należy ustawić wartość zmiennej *ApiBaseAddress* w klasie *WebRepositoryBase* znajdującej się w projekcie *Core* w przestrzeni nazw *Repositories.Web*. Przechowuje ona adres pod jakim znajduje się uruchomiona instancja części serwerowej systemu. Pod zdefiniowany adres będą więc kierowane wszystkie zapytania HTTP wychodzące z aplikacji mobilnej. Przykładowo zdefiniowany adres przedstawiono na listingu 6.2.

```
public abstract class WebRepositoryBase
{
      private const string ApiBaseAddress =
            "http://192.168.1.16:5000/";
      protected readonly HttpClient Client;

      protected WebRepositoryBase()
      {
            Client = new HttpClient { BaseAddress =
                  new Uri(ApiBaseAddress) };
      }
}
```

Listing 6.2: Klasa zawierająca adres aplikacji serwerowej

Po wykonaniu tej czynności należy pobrać zależności i zbudować aplikację za pomocą polecenia *dotnet build*.

6.2. Warstwa modelu danych

Informacje o trasach i wynikach przesyłane są z aplikacji mobilnej do serwerowej, a więc klasy reprezentujące model danych w obu projektach pokrywają się w znacznej części.

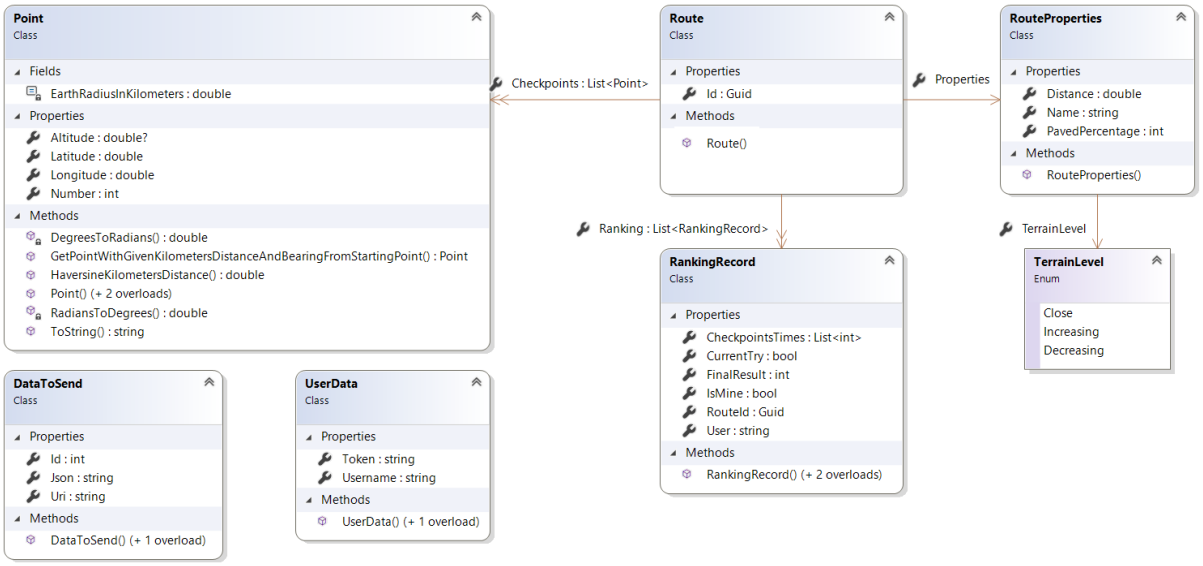
6.2.1. Aplikacja mobilna

Zaprojektowany model danych zgodnie z założeniami pozwala na wyznaczenie oraz przechowywanie cech charakteryzujących trasę. Jego reprezentacja w formie diagramu klas została przedstawiona na rysunku 6.1. Pierwszoplanową rolę w modelu danych odgrywa klasa *Route*, będąca główną reprezentacją trasy. Sama w sobie nie przechowuje ona żadnych danych z wyjątkiem identyfikatora trasy w systemie, jednak zawiera odwołania do pozostałych klas:

- *Point* - Przechowuje informacje o pojedynczym punkcie z którego składa się trasa. Warto zauważyć, że typ danych właściwości (ang. property) pozwala na przypisanie wartości *null*. Spowodowane jest to faktem, iż istnieje prawdopodobieństwo posiadania informacji o szerokości i długości geograficznej nie znając jednocześnie wysokości nad poziomem morza. Zjawisko to zostało opisane w rozdziale 2.3.1. Do każdego punktu przypisany jest także jego numer oznaczający kolejność w której występuje na trasie.
- *RouteProperties* - Przechowuje nazwę oraz cechy trasy, a więc dystans, poziom twardości nawierzchni oraz poziom nachylenia terenu będący typem wyliczeniowym.
- *RankingRecord* - Przechowuje informacje o próbach użytkowników na trasie. Czasy osiągane na poszczególnych punktach kontrolnych zapisane są we właściwości *CheckpointTimes* będącej kolekcją. Jej *i-ty* element przechowuje liczbę sekund, która upłynęła od rozpoczęcia treningu w momencie „osiągnięcia” *i-tego* punktu kontrolnego. Oprócz tego w klasie znajdują się właściwości identyfikujące trasę oraz użytkownika oraz pozwalające stwierdzić czy konkretna próba na trasie należy do aktualnie zalogowanego użytkownika (*IsMine*) oraz czy oznacza aktualnie trwającą próbę (*IsCurrentTry*).

Ponad to model danych zawiera dwie dodatkowe klasy:

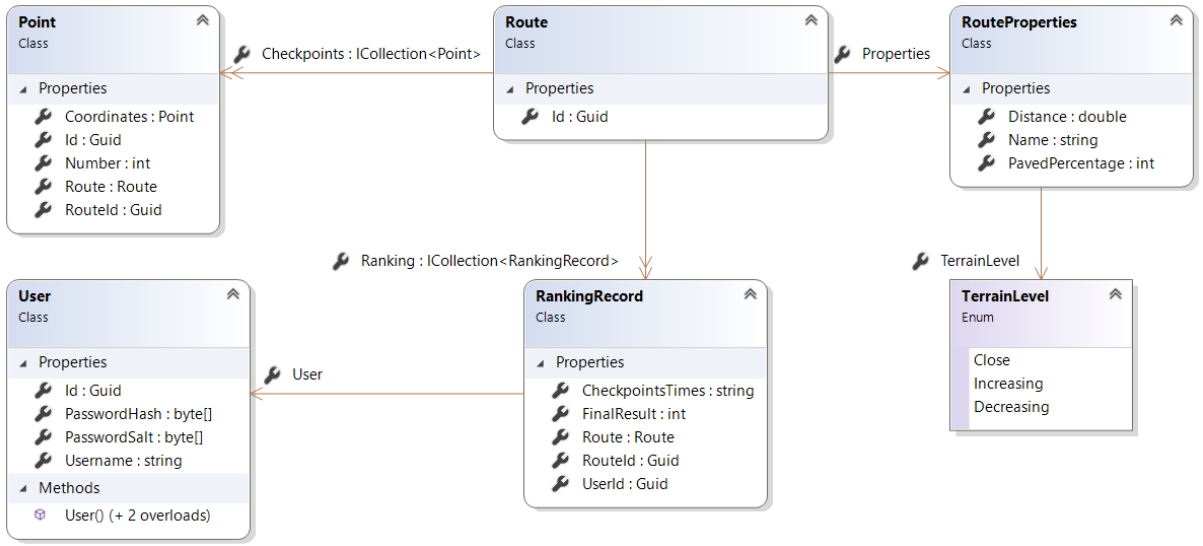
- *DataToSend* - przechowuje dane w formacie JSON oraz adres URL. Pozwala podjąć ponowną, odłożoną w czasie próbę wysłania danych na serwer w przypadku gdy proces ten nie zakończy się sukcesem przy pierwszym podejściu.
- *UserData* - przechowuje dane o aktualnie zalogowanym użytkowniku: login użytkownika oraz token używany podczas komunikacji z serwerem.



Rysunek 6.1: Diagram klas modelu danych aplikacji mobilnej [Opracowanie własne]

6.2.2. Aplikacja serwerowa

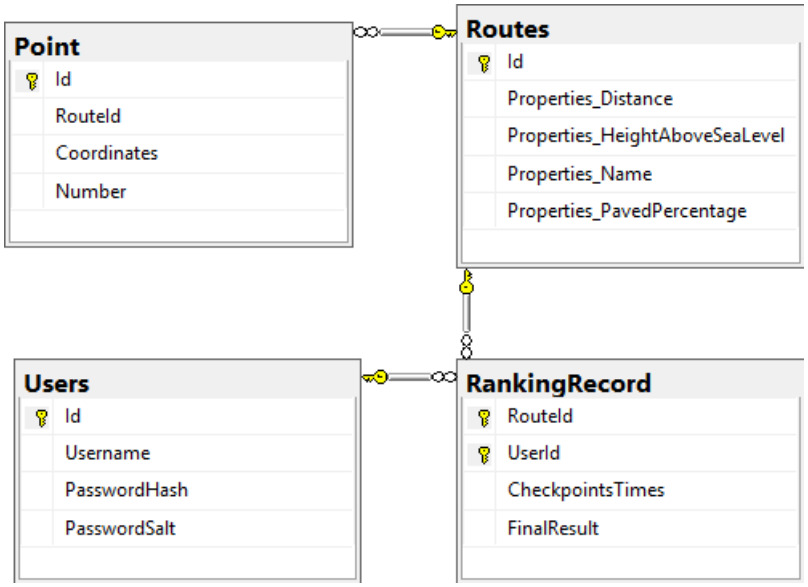
Z racji podobieństwa do modelu danych zawartego w aplikacji mobilnej, który opisany został w rozdziale 6.2.1, w rozdziale niniejszym zostaną przybliżone jedynie nie omówione wcześniej aspekty.



Rysunek 6.2: Diagram klas modelu danych aplikacji serwerowej [Opracowanie własne]

Jak widać na diagramie 6.2 w modelu aplikacji serwerowej istnieje klasa *User*. Reprezentuje ona konto użytkownika w systemie. Powiązanie tej klasy z klasą reprezentującą próbę na trasie, pozwala określić do którego z użytkowników należy konkretny wynik.

Współrzędne punktów trasy zapisywane przy użyciu typu danych *Point*. Należy zaznaczyć, że nie jest to typ stworzony w ramach modelu danych, lecz należącą do przestrzeni nazw *NetTopologySuite.Geometries*, występującą w środowisku .NET reprezentacją typu *geography* istniejącego w systemie zarządzania bazą danych SQL Server. *Geography* pozwala na przechowywanie współrzędnych geograficznych punktu (wraz z wysokością nad poziomem morza) w jednej kolumnie tabeli bazodanowej. [**?**]



Rysunek 6.3: Diagram encji bazy danych [Opracowanie własne]

Zastosowanie mapowania obiektowo-relacyjnego pozwoliło na odwzorowanie modelu danych w formie bazy danych. Jej struktura została przedstawiona na rysunku 6.3. Przed utworzeniem odpowiednich tabel konieczne było zdefiniowanie relacji pomiędzy polami tabel. Odpowiednią konfigurację umieszczono w klasie *Context* znajdującej się w przestrzeni nazw *Api.Entities* i przedstawiono na listingu 6.3.

```
1  protected override void OnModelCreating(
2  ModelBuilder modelBuilder)
3  {
4      modelBuilder.Entity<Route>().HasMany(r => r.Checkpoints)
5          .WithOne(cp => cp.Route)
6          .HasForeignKey(cp => cp.RouteId);
7      modelBuilder.Entity<Route>().OwnsOne(r => r.Properties);
8      modelBuilder.Entity<Route>().HasMany(r => r.Ranking)
9          .WithOne(rr => rr.Route)
10         .HasForeignKey(rr => rr.RouteId);
```

```
11
12      modelBuilder.Entity<RankingRecord>().HasOne(rr => rr.User);
13      modelBuilder.Entity<RankingRecord>()
14          .HasKey(rr => new { rr.RouteId, rr.UserId });
15
16      base.OnModelCreating(modelBuilder);
17  }
```

Listing 6.3: Konfiguracja mapowania relacyjno-obiektowego

Konfiguracja pozwoliła na utworzenie następujących relacji:

- jeden do wielu pomiędzy tabelami reprezentującymi klasy *Route* oraz *Point*
- jeden do wielu pomiędzy tabelami reprezentującymi klasy *Route* oraz *RankingRecord*
- jeden do jednego pomiędzy tabelami reprezentującymi klasy *RankingRecord* oraz *User*

Dzięki instrukcji z linii numer 7 dane z klas *Route* oraz *RouteProperties* zostały umieszczone w jednej tabeli. Pozwoliło to na uniknięcie łączenia tabel podczas pobierania danych. Typy wyliczeniowe nie wymagają osobnej tabeli i są przechowywane w postaci liczb, dlatego nie było potrzeby tworzenia dodatkowej struktury dla cechy *poziom terenu*.

Warto zauważyć, że czasy osiągane na punktach kontrolnych są przechowywane jako łańcuch znaków (*string*). Pozwoliło to uniknąć tworzenia dodatkowej tabeli i relacji jeden do wielu z tabelą reprezentującą klasę *RankingRecord*. Mapowanie pomiędzy kolekcją a łańcuchem znaków odbywa się w klasie *AutomapperConfig* znajdującej się w przestrzeni nazw *Api.Mappers* i zostało przedstawione na listingu **??**.

```
1  public static IMapper Initialize()
2      => new MapperConfiguration(cfg =>
3          {
4           cfg.CreateMap<RankingRecord, RankingRecordDto>()
5              .ForMember(dest => dest.CheckpointsTimes, opt =>
6                   opt.MapFrom(src => src.CheckpointsTimes
7                   .Split(" ", StringSplitOptions.None).Select(int.Parse)))
8           cfg.CreateMap<RankingRecordDto, RankingRecord>()
9              .ForMember(dest => dest.CheckpointsTimes,
10                 opt => opt.MapFrom(src =>
11                 string.Join(" ", src.CheckpointsTimes)))
12         })
13         .CreateMapper();
```

Listing 6.4: Mapowanie pomiedzy kolekcją a łańcuchem znaków

6.3. Warstwa logiki biznesowej

6.3.1. Wyszukiwanie tras

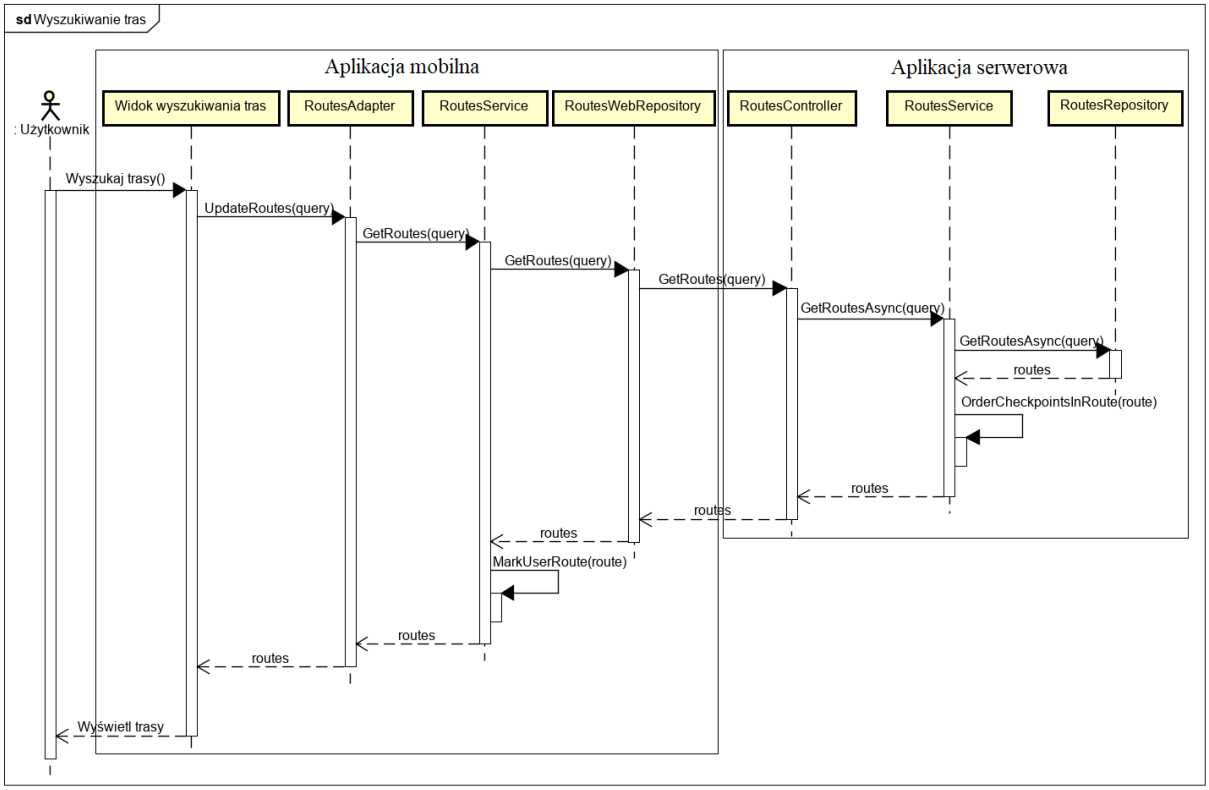
Wyszukiwanie tras odbywa się poprzez wysłanie zapytania z aplikacji mobilnej do aplikacji serwerowej. Ustalone przez użytkownika kryteria wyszukiwania są następnie przekazywane do bazy danych. Dzięki temu proces odfiltrowania tras ma miejsce tak wcześnie jak to możliwe i do aplikacji serwerowej, a następnie mobilnej trafiają tylko te trasy, które rzeczywiście spełniają ustalone kryteria. Diagram sekwencji tego procesu został pokazany na rysunku 6.4. Kryteria wybrane przez użytkownika zapisane są w instancji klasy *RoutesFilterQuery*, która pozwala zachować wszystkie informacje opisane w rozdziale 3.3. Obiekt ten jest przekazywany do części serwerowej systemu w celu dokonania odpowiedniej filtracji tras.

W momencie gdy zostanie wywołana metoda *GetRoutesAsync* klasy *RoutesRepository*, przekazany obiekt jest wykorzystany do określenia warunków, które muszą spełniać pobierane trasy. Działanie to zostało pokazane na listingu 6.5. Warto podkreślić, że dzięki zastosowaniu typu *geography* w tabeli bazy danych, możliwe jest policzenie odległości pomiędzy pozycją użytkownika a początkiem trasy bezpośrednio na poziomie bazy danych [**?**, **?**]. Warunek ten określa linia numer 8 listingu. Na podstawie określonych warunków generowane jest następnie zapytanie SQL wykonywane bezpośrednio na poziomie bazy danych.

```
1  var routes = _context.Routes
2          .Where(r => r.Properties.PavedPercentage
3          >= query.SurfacePavedPercentageFrom
4         && r.Properties.PavedPercentage <= query.SurfacePavedPercentageTo)
5         .Where(r => r.Properties.Distance >= query.RouteLengthFrom
6         && r.Properties.Distance <= query.RouteLengthTo)
7         .Where(r => r.Checkpoints.First(cp => cp.Number == 0)
8         .Coordinates.IsWithinDistance(currentLocation,
9         query.SearchRadiusInMeters));
10
11 if (query.SurfaceLevel > 0)
12         routes = routes.Where(r => r.Properties.TerrainLevel
```

```
13          == (TerrainLevel)query.SurfaceLevel);
```

Listing 6.5: Określenie warunków dla pobieranych tras



Rysunek 6.4: Diagram sekwencji procesu wyszukiwania tras [Opracowanie własne]

Po pobraniu danych z bazy danych, w klasie *RoutesService* z przestrzeni nazw *Api.Services* aplikacji serwerowej punkty kontrolne każdej trasy są sortowane według pola *Number*, tak aby ich kolejność w kolekcji odpowiadała kolejności, którą powinny zajmować na trasie. Po przesłaniu danych o trasach z aplikacji serwerowej do aplikacji mobilnej, w klasie *RoutesService* znajdującej się przestrzeni nazw *Core.Services*, odbywa się aktualizacja własciwości *IsMine*. Proces ten został pokazany na listingu 6.6. Dzięki temu w interfejsie użytkownika możliwe jest wyróżnienie wyniku, który należy do biegacza.

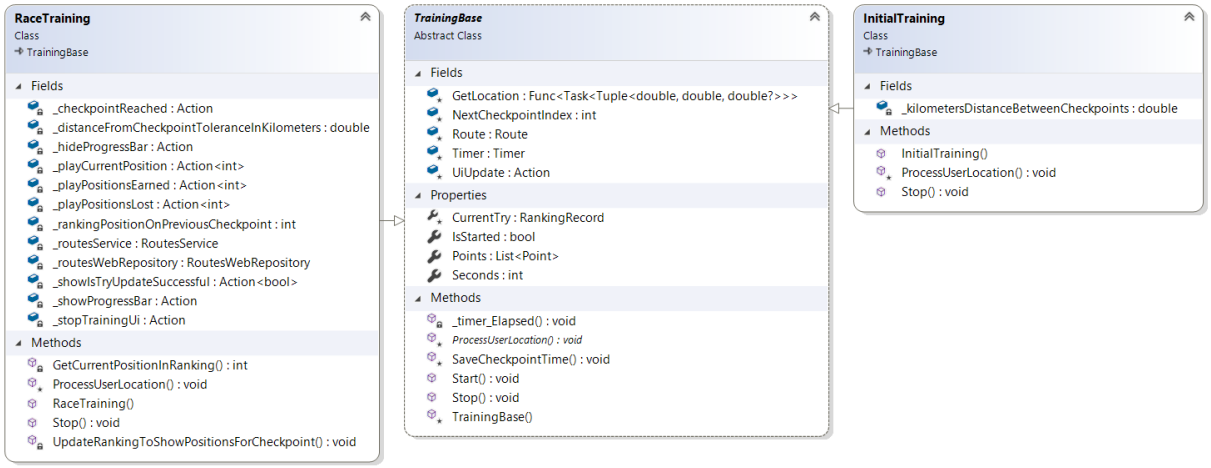
```
1  private void MarkUserRoute(Route route)
2  {
3          var mineRankingRecord = route.Ranking
4          .FirstOrDefault(r => r.User == _userData.Username);
5          if (mineRankingRecord != null)
6                mineRankingRecord.IsMine = true;
7  }
```

Listing 6.6: Aktualizacja właściwości IsMine klasy Route

6.3.2. Odbywanie treningu

Implementacje treningu, który tworzy nową trasę jak i trening przeprowadzany w ramach istniejącej trasy, który zawiera w sobie dodatkowo element wirtualnej rywalizacji zawierają wspólną logikę, dlatego tak jak widać na rysunku 6.5, skorzystano z mechanizmu dziedziczenia. Należy zwrócić uwagę, że klasy te zawierają kilka pól typu *Action*. Dzięki nim możliwe jest aktualizowanie interfejsu użytkownika z poziomu metody klasy należącej do logiki biznesowej. W przypadku stworzenia aplikacji mobilnej przeznaczonej na inny system operacyjny, wystarczy przekazać odpowiednie implementacje metod odpowiedzialnych za aktualizację interfejsu użytkownika, a zostaną one wywołane.

Metody *Start* oraz *Stop* służą do odpowiednio do rozpoczęcia i zakończenia treningu. Zawierają one logikę odpowiedzialną za dodanie obecnej próby użytkownika do rankingu oraz uruchomienie i zatrzymanie stopera. Dzięki stoperowi za każdym razem gdy upłynie jedna sekunda, aktualizowany jest interfejs użytkownika oraz wywoływana jest metoda *ProcessUserLocation*, której implementacja zależna jest od typu treningu.



Rysunek 6.5: Diagram klas odpowiadających za przeprowadzanie treningów [Opracowanie własne]

Trening tworzący trasę

Za logikę obsługującą trening na nowej trasie odpowiada klasa *InitialTraining*. Za każdym razem gdy wywołana zostanie metoda *ProcessUserLocation*, podejmowana jest próba dodania

do tworzonej trasy nowego punktu kontrolnego na podstawie obecnej pozycji biegacza. Aby punkt mógł zostać dodany, użytkownik musi znajdować się w pozycji nie mniejszej niż 10 metrów od ostatnio utworzonego punktu kontrolnego. Prawdziwość tego warunku jest sprawdzana w linii numer 6, 7 i 8 na listingu 6.8. Działanie ma to na celu uniknięcie problemu opisanego w rozdziale 2.3.2, którego wystąpienie spowodowałoby utworzenie skupiska punktów kontrolnych oddalonych od siebie o kilka metrów. Wyznaczenie odległości pomiędzy aktualną pozycją użytkownika, a ostatnio utworzonym punktem kontrolnym obliczana jest za pomocą formuły Haversine przytoczonej w rozdziale 2.2.1. Jej implementacja została pokazana na listingu 6.7. Współrzędne odczytywane są w stopniach, dlatego przed obliczeniem odległości, konieczne było wyznaczenie ich wartości w radianach.

```
1  public static double HaversineKilometersDistance(Point point1,
2        Point point2)
3  {
4    var latitudeDelta = DegreesToRadians(point1.Latitude - point2.Latitude);
5    var longitudeDelta = DegreesToRadians(
6        point1.Longitude - point2.Longitude);
7
8    var firstPointLatitudeCos = Math.Cos(DegreesToRadians(point1.Latitude));
9    var secondPointLatitudeCos = Math.Cos(
10       DegreesToRadians(point2.Latitude));
11
12   var a = Math.Pow(Math.Sin(latitudeDelta / 2), 2)
13       + firstPointLatitudeCos *
14       secondPointLatitudeCos * Math.Pow(Math.Sin(longitudeDelta / 2), 2);
15   var c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
16   var d = EarthRadiusInKilometers * c;
17
18   return d;
19 }
```

Listing 6.7: Wyznaczenie odległości pomiędzy dwoma punktami

```
1  protected override async void ProcessUserLocation()
2  {
3     var location = await GetLocation();
4     var currentPosition = new Point(
5        location.Item1, location.Item2,location.Item3,NextCheckpointIndex);
```

```
6     var distanceFromLastPoint = Route.Checkpoints.Any()
7         ? Point.HaversineKilometersDistance(Route.Checkpoints.Last(),
8         currentPosition)
9         : _kilometersDistanceBetweenCheckpoints;
10    if (distanceFromLastPoint >= _kilometersDistanceBetweenCheckpoints)
11    {
12        NextCheckpointIndex++;
13        Route.Checkpoints.Add(currentPosition);
14        SaveCheckpointTime();
15    }
16 }
```

Listing 6.8: Tworzenie punktów kontrolnych

Wraz z utworzeniem punktu kontrolnego, zapisywany jest także czas, który upłynął od rozpoczęcia treningu. Oznacza to, że biegacz tworzący trasę, zostaje jednocześnie pierwszym użytkownikiem dodanym do rankingu.

W momencie gdy użytkownik postanawia zakończyć trening, wywołana zostaje metoda *Stop*. Oprócz bazowej implementacji tej metody, zostaje także wykonany kod odpowiedzialny za utworzenie ostatniego punktu kontrolnego. Jest on tworzony niezależnie od odległości do przedostatniego punktu kontrolnego, a jego pozycja odpowiada pozycji użytkownika z momentu zakończenia treningu.

Trening na zapisanej trasie oraz wirtualna rywalizacja

Proces wirtualnej rywalizacji odbywa się wraz z rozpoczęciem treningu na stworzonej wcześniej trasie. Za obsługę obu tych elementów odpowiada klasa *RaceTraining*. Metoda *ProcessUserLocation* służy do zapisywania czasów, które biegacz osiągnął na każdym z punktów kontrolnych. Jej kod źródłowy został przedstawiony na listingu 6.9.

```
1 protected override async void ProcessUserLocation()
2 {
3   var location = await GetLocation();
4   var currentLocation = (new Point(location.Item1,
5     location.Item2, Seconds));
6   var distance = Point.HaversineKilometersDistance(currentLocation,
7     Route.Checkpoints[NextCheckpointIndex]);
8
9   if (distance < _distanceFromCheckpointToleranceInKilometers)
```

```
10  {
11      NextCheckpointIndex++;
12
13      SaveCheckpointTime();
14      if (NextCheckpointIndex == Route.Checkpoints.Count)
15      {
16        Stop();
17        _routesService.ProcessCurrentTry(Route, CurrentTry);
18
19        _stopTrainingUi.Invoke();
20
21        _showProgressBar.Invoke();
22        var isSuccessful = await _routesWebRepository
23              .CreateRankingRecordAsync(CurrentTry, Route.Id);
24        _hideProgressBar.Invoke();
25        _showIsTryUpdateSuccessful.Invoke(isSuccessful);
26      }
27      else
28      {
29        UpdateRankingToShowPositionsForCheckpoint(NextCheckpointIndex - 1);
30
31        var currentPosition = GetCurrentPositionInRanking();
32        if (currentPosition ==
33          rankingPositionOnPreviousCheckpoint
34            || _rankingPositionOnPreviousCheckpoint == 0)
35        {
36           _playCurrentPosition.Invoke(currentPosition);
37        }
38        else
39        {
40           if (currentPosition < _rankingPositionOnPreviousCheckpoint)
41              _playPositionsEarned(
42              _rankingPositionOnPreviousCheckpoint
43                - currentPosition);
44           else
45              _playPositionsLost(currentPosition
46                - _rankingPositionOnPreviousCheckpoint);
47         }
48
```

```
49          _rankingPositionOnPreviousCheckpoint = currentPosition;
50
51          _checkpointReached.Invoke();
52        }
53    }
54  }
```

Listing 6.9: Proces wirtualnej rywalizacji

Biegacz musi znajdować się w odległości nie większej niż 10 metrów od następnego w kolejności punktu kontrolnego. Gdy ten warunek zostanie spełniony, bieżący czas od rozpoczęcia treningu zostaje zapisany. Odległość pomiędzy użytkownikiem a punktem kontrolnym wyznaczana jest za pomocą formuły haversine, której implementacja znajduje się na listingu 6.7. Za każdym razem gdy punkt kontrolny zostanie osiągnięty, aktualizowany jest ranking. Odbywa się to poprzez posortowanie kolekcji *Ranking* znajdującej się w instancji klasy *Route* względem wartości, która reprezentuje czas osiągnięty przez wszystkich użytkowników na punkcie kontrolnym. Oprócz faktu, że pozycje w rankingu wyświetlonym na ekranie urządzenia zmieniają się wraz z trwaniem biegu, odtwarzane są komunikaty głosowe. Logika odpowiadająca za ten proces znajduje się w liniach 32-47 listingu 6.9 i jest implementacją warunków opisanych w rozdziale 4.2.

W momencie gdy zostaną osiągnięte wszystkie punkty kontrolne, trening jest automatycznie zakańczany, a próba użytkownika jest poddawana przetworzeniu. Scenariusz ten realizują linie 15-26 listingu 6.9. Proces przetworzenia próby jest implementacją warunków opisanych w rozdziale 4.1 i został przedstawiony na listingu 6.10.

```
1  public void ProcessCurrentTry(Route route, RankingRecord currentTry)
2  {
3     var lastTry = route.Ranking.SingleOrDefault(rr => rr.IsMine);
4     if (lastTry == null)
5        return;
6     if (currentTry.FinalResult < lastTry.FinalResult)
7     {
8        currentTry.IsMine = true;
9        currentTry.IsCurrentTry = false;
10       currentTry.User = lastTry.User;
11
12       route.Ranking.Remove(lastTry);
```

```
13      }
14      else
15          route.Ranking.Remove(currentTry);
16  }
```

Listing 6.10: Przetworzenie próby użytkownika

Logika ta jest wykonywana na urządzeniu mobilnym, dzięki czemu użytkownik ma możliwość sprawdzenia końcowego rankingu wirtualnej rywalizacji. Warto zaznaczyć, że próba jest także wysyłana do aplikacji serwerowej, gdzie przetwarzana jest w ten sam sposób. Ma to miejsce nawet w przypadku, gdy końcowa lokata użytkownika w rankingu na urządzeniu mobilnym nie zmieniła się. Jest to spowodowane brakiem gwarancji, że ranking zapisany w bazie danych, jest nadal w tym samym stanie, co w momencie pobrania go na urządzenie mobilne przed rozpoczęciem treningu.

6.3.3. Automatyczne przypisywanie cech do trasy

Logikę odpowiedzialną za automatyczne przypisywanie cech umieszczono w klasie *RoutesService*, która znajduje się w przestrzeni nazw *Core.Services*. Została zaimplementowana zgodnie z opisem zawartym w rozdziale 3.2.

Długość trasy

Wyznaczenie długości polega na zsumowaniu odległości pomiędzy wszystkimi punktami trasy przy pomocy formuły haversine, której implementację pokazano na listingu 6.7.

```
1  public void CalculateRouteDistance(Route route)
2  {
3      var totalDistance = 0d;
4      for (int i = 0; i < route.Checkpoints.Count - 1; i++)
5      {
6          totalDistance += Point.HaversineKilometersDistance(
7                route.Checkpoints[i], route.Checkpoints[i + 1]);
8      }
9      route.Properties.Distance = Math.Round(totalDistance, 2);
10 }
```

Listing 6.11: Wyznaczenie długości trasy

Poziom terenu

Wyznaczenie poziomu terenu zostało pokazane na listingu 6.12. Wartość zmiennej *terrainLevelDifferenceThresholdInPercentage* wyraża maksymalną procentową różnicę pomiędzy wysokością początkową a końcową dla której poziom terenu będzie uznany jako wyrównany. W tym przypadku wynosi ona 10%.

Ponadto poziom terenu zostanie wyznaczony jako „wyrównany", gdy nie udało się wyznaczyć wysokości któregokolwiek z punktów.

```
1  private void ResolveTerrainLevel(Route route)
2  {
3     const int terrainLevelDifferenceThresholdInPercentage = 10;
4     const double terrainLevelDifferenceThreshold =
5        1.0 * terrainLevelDifferenceThresholdInPercentage / 100 + 1;
6     var startAltitude = route.Checkpoints.First().Altitude;
7     var finishAltitude = route.Checkpoints.Last().Altitude;
8
9     route.Properties.TerrainLevel = TerrainLevel.Close;
10    if (startAltitude != null && finishAltitude != null)
11    {
12       if (startAltitude
13             > finishAltitude * terrainLevelDifferenceThreshold)
14       {
15          route.Properties.TerrainLevel = TerrainLevel.Decreasing;
16       }
17       else if (finishAltitude
18             > startAltitude * terrainLevelDifferenceThreshold)
19       {
20          route.Properties.TerrainLevel = TerrainLevel.Increasing;
21       }
22    }
23 }
```

Listing 6.12: Wyznaczenie poziomu terenu

Twardość nawierzchni

Za wyznaczenie tej cechy odpowiada klasa *OsmService*. Główna metoda odpowiedzialna za to działanie została przedstawiona na listingu 6.13. Metoda wywoływana w linii numer 3

listingu zajmuje się zbudowaniem odpowiedniego zapytania zgodnie z zasadami opisanymi w [**?**], wysłaniem go do serwisu OpenStreetMap [**?**] oraz zwróceniem otrzymanej odpowiedzi. W efekcie do zmiennej *tags* zostają przypisane rodzaje nawierzchni dla każdego punktu trasy, natomiast zmienne *PavedSurfaces* oraz *UnpavedSurfaces* zawierają wszystkie możliwe utwardzone oraz nieutwardzone rodzaje nawierzchni zgodnie z tabelą 3.1 z rozdziału 3.2.3. Na tej podstawie w liniach 4-7 wyznaczana jest procentowa twardość nawierzchni utworzonej trasy. W przypadku gdy nie uda się ustalić twardość nawierzchni (na przykład gdy z zewnętrznego serwisu nie uda się otrzymać żadnych informacji) twardość trasy zostaje ustalona jako wartość zmiennej *DefaultSurfacePavement* wynosząca 50%.

```
1  public async Task<int> ResolveRouteSurfaceTypeAsync(Route route)
2  {
3    var tags = (await GetSurfaceTypesAsync(route.Checkpoints)).ToList();
4    int pavedCount = tags.Count(t => PavedSurfaces.Contains(t));
5    int unpavedCount = tags.Count(t => UnpavedSurfaces.Contains(t));
6
7    var pavedPercent = 1.0 * pavedCount / (pavedCount + unpavedCount) * 100;
8    if (double.IsNaN(pavedPercent))
9       return DefaultSurfacePavement;
10
11   return (int)Math.Round(pavedPercent);
12 }
```

Listing 6.13: Wyznaczenie poziomu terenu

6.3.4. Brak połączenia z serwerem po zakończonym treningu

Może zdarzyć się, że nie uda się zapisać treningu użytkownika, ponieważ urządzenie nie miało dostępu do internetu lub nastąpiła awaria serwera. W takim wypadku użytkownik utraciłby swoją próbę. Aby tego uniknąć, w momencie gdy nie uda się nawiązać połączenia z serwerem, dane o treningu w formacie JSON oraz adres pod który miały być wysłane zapisywane są w lokalnej bazie danych urządzenia. Następnie możliwe jest podjęcie ponownej próby wysłania zaległych danych. Proces ten został przedstawiony na listingu 6.14. Metoda *ProcessOverdueData* wywoływana jest przy każdym uruchomieniu aplikacji. Z lokalnej bazy danych pobierane są wszystkie zaległe treningi, a następnie po raz kolejny wysłane do

części serwerowej aplikacji w metodzie wywoływanej w linii numer 6 listingu. W przypadku powodzenia tej operacji trening jest usuwany z lokalnej bazy danych.

```
1  public async Task ProcessOverdueData()
2  {
3     var dataToSend = _userLocalRepository.GetDataToSend();
4     foreach (var data in dataToSend)
5     {
6        var result = await _routesWebRepository
7              .SendJsonData(data.Json, data.Uri);
8
9        if (result)
10          _userLocalRepository.DeleteDataToSend(data.Id);
11    }
12 }
```

Listing 6.14: Wyznaczenie poziomu terenu

6.4. Warstwa interfejsu użytkownika

6.4.1. Internacjonalizacja

7. Dokumentacja użytkownika

8. Podsumowanie

Bibliografia

[1] Janusz Narkiewicz. *GPS i inne satelitarne systemy nawigacyjne, wyd. 1*. Wydawnictwa Komunikacji i Łączności, 2007. (Cytowanie na stronach 7, 9 i 10.)

Spis rysunków

Spis tabel

Spis listingów