

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Runly: Running App

Autor:

Łukasz Skraba
Bartosz Tobiasz

Prowadzący:

mgr inż. Dawid Kotlarski

Nowy Sącz 2022

Spis treści

1. Ogólne określenie wymagań	4
1.1. Opis działania	4
1.2. Opis wyglądu	4
2. Określenie wymagań szczegółowych	6
2.1. Środowisko programistyczne	6
2.2. Ogólny wygląd interfejsu	7
2.3. Krokomierz	7
2.4. Moduł GPS	7
2.5. Mapy Google	8
2.6. Baza danych SQLite	8
3. Projektowanie	9
3.1. Przygotowanie narzędzi (Git, Visual Studio)	9
3.1.1. Visual Studio	9
3.1.2. Git	9
3.2. Szkice layoutów	10
3.2.1. Zakładka Kroki	10
3.2.2. Zakładka Trening	10
3.2.3. Zakładka Historia	11
3.2.4. Zakładka Ustawienia	12
4. Implementacja	13
4.1. Wybrane kody z objaśnieniami działania	13
4.1.1. Zakładka Historia	13
4.1.2. Zakładka Trening	15
4.1.3. Zakładka Kroki	20
5. Testowanie	23
6. Podręcznik użytkownika	24
Literatura	25

Spis rysunków	26
Spis tabel	27
Spis listingów	28

1. Ogólne określenie wymagań

1.1. Opis działania

Aplikacja na urządzenia mobilne umożliwiająca monitoring dokonań sportowych w dziedzinie biegania. Program ma umożliwić monitorowanie naszej aktywności biegowej. Aplikacja ma zapisywać przede wszystkim czas treningu, dystans, trasę uzyskaną dzięki modułowi GPS oraz intensywność treningu (np. wyliczając średnie tempo, średnią i maksymalną prędkość oraz spalone kalorie). Korzystając z aplikacji mamy mieć możliwość szczegółowej weryfikacji danych treningu, zarówno w trakcie jego trwania jak i po jego zakończeniu. Dodatkowo w podsumowaniu dzięki współpracy programu z GPS-em, można także sprawdzić informacje o najniższym i najwyższym punkcie trasy. Szczegółowe statystyki mają pozwolić na analizę postępów i wyciągnięcie wniosków na przyszłość.

Treningi mają być zapisywane w pamięci. Użytkownik ma mieć możliwość zobaczenia statystyk wybranego treningu.

Aplikacja ma za zadanie także motywować nas do ćwiczeń, np. wysyłając nam powiadomienia, w ustalonym przez użytkownika momencie, o tym, że nie odbyliśmy jeszcze treningu.

Poza pomiarami w trakcie treningu, aplikacja ma także liczyć kroki, kiedy działa w tle.

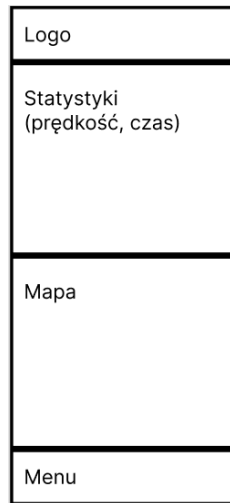
1.2. Opis wyglądu

Na głównej stronie treningu, którą widzi użytkownik po otwarciu aplikacji, powinny znajdować się takie informacje jak:

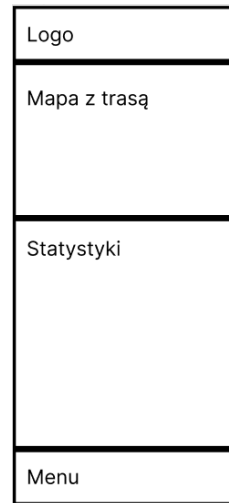
- Czas trwania aktywności,
- Prędkość w danym momencie,
- Średnia prędkość,
- Dystans,
- Spalone kalorie

Oprócz tego na stronie treningu Rys. 1.1 (s. 5) powinna znajdować się mapa, na której będzie pokazana aktualna pozycja użytkownika oraz przebyta trasa.

Po zakończonym treningu aplikacja ma pokazać całą przebytą trasę na mapie oraz dać dostęp do szczegółowych statystyk treningu Rys. 1.2 (s. 5). Użytkownik ma mieć podgląd na wszystkie możliwe dane.

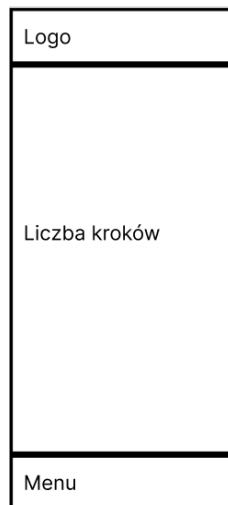


Rys. 1.1. Ekran treningu



Rys. 1.2. Ekran podsumowania

Ekran krokomierza Rys. 1.3 (s. 5) ma zawierać tylko liczbę zrobionych w bieżącym dniu kroków.



Rys. 1.3. Ekran krokomierza

Na dole aplikacji ma się znajdować menu za pomocą którego użytkownik może się przełączać pomiędzy ekranem krokomierza, treningu, odbytymi treningami i ustawieniami.

Ekran zawierający historię odbytych treningów powinien przedstawiać je w postaci list. Ustawienia też powinny być przedstawione w postaci listy.

2. Określenie wymagań szczegółowych

2.1. Środowisko programistyczne

Aplikacja zostanie napisana korzystając z platformy Xamarin.Forms. Jest to środowisko umożliwiające tworzenie aplikacji za pomocą języka XAML oraz kodu w języku C#. Xamarin to platforma typu open source do tworzenia nowoczesnych i wydajnych aplikacji dla systemów iOS, Android i Windows za pomocą platformy .NET. Xamarin to warstwa abstrakcji, która zarządza komunikacją udostępnionego kodu z bazowym kodem platformy. Xamarin.Forms umożliwia deweloperom tworzenie aplikacji Xamarin.iOS, Xamarin.Android i Windows z pojedynczej udostępnionej bazy kodu. Xamarin.Forms umożliwia deweloperom tworzenie interfejsów użytkownika w języku XAML z użyciem kodu w języku C#. Te interfejsy użytkownika są renderowane jako wydajne kontrolki natywne na każdej platformie.

Oto kilka przykładów funkcji udostępnianych przez Xamarin.Forms:

- Język interfejsu użytkownika XAML,
- Powiązanie danych,
- Gestów,
- Efekty,
- Style

Dodatkowo, w celu koordynacji wykonywania projektu, organizacji kodu oraz ogólnego wspomagania pracy wykorzystany zostanie system kontroli wersji oprogramowania Git. Zapewnia on:

- Dobre wsparcie dla rozgałęzionego procesu tworzenia oprogramowania: jest dostępnych kilka algorytmów łączenia zmian z dwóch gałęzi, a także możliwość dodawania własnych algorytmów,
- Każdy programista posiada własną kopię repozytorium, do której może zapisywać zmiany bez połączenia z siecią; następnie zmiany mogą być wymieniane między lokalnymi repozytoriami. Programista może także dodawać oraz usuwać gałęzie,
- Efektywną pracę z dużymi projektami, jest o rzędy wielkości szybszy niż niektóre konkurencyjne rozwiązania

2.2. Ogólny wygląd interfejsu

Aplikacja będzie podzielona na podstrony. Na dole będzie znajdować się menu z opcjami. Wybór poszczególnej opcji w menu wyświetli daną podstronę w ekranie aplikacji. Kolejne podstrony to: Kroki, Trening, Historia, Ustawienia.

2.3. Krokomierz

W zakładce Kroki aplikacja będzie wyświetlała, korzystając z odpowiedniego sensora (krokomierza), liczbę kroków jaką użytkownik wykonał od rozpoczęcia treningu. Moduł krokomierza jest wymagany by zwiększyć wiarygodność aplikacji. Zastosowanie krokomierza umożliwia wykluczenie wpisów podejrzanych, które mogłyby wystąpić, gdyby użytkownik jechał na przykład rowerem lub samochodem. Takie rozwiązanie jest normalne w tego typu aplikacjach. Oprogramowanie Android w wersjach 4.4 i wyższych posiada wsparcie dla sensorów takich jak detektor kroków oraz licznik kroków. Kroki podczas biegu są łatwiejsze do odróżnienia od tych podczas zwykłego spaceru ze względu na bardziej wyraźne oddziaływanie na sensory. Tak więc jest duże prawdopodobieństwo, że krokomierz będzie bardzo dokładnie mierzył kroki, a system będzie działał niezawodnie w wyjątkowych sytuacjach.

2.4. Moduł GPS

W zakładce Trening aplikacja będzie wyświetlała pomiar biegu użytkownika, a dokładniej mówiąc pomiar przebiegniętego dystansu, prędkości biegu w danej chwili oraz czasu. Odbywa się to na podstawie informacji o jego pozycji. Można to zrealizować na kilka sposobów. Te sposoby to: wykorzystanie globalnego systemu pozycjonowania (GPS), technologia lokalizacji wieży komórkowej lub lokalizacja za pomocą WiFi. Powinno się wybrać sposób najbardziej odpowiedni dla naszej aplikacji biorąc pod uwagę przede wszystkim środowisko w jakim będzie ona używana. W naszym przypadku jest to system GPS, gdyż zapewnia on najdokładniejsze dane lokalizacyjne, wykorzystuje najwięcej mocy i działa najlepiej na zewnątrz, co w naszym przypadku jest kluczowe dla odpowiedniego działania aplikacji. Zakładając, że użytkownik przemieszcza się, można zdefiniować jego pozycję z dokładnością od około 6 do 100 metrów.

Aplikacja z obsługą lokalizacji wymaga dostępu do czujników sprzętowych urządzenia w celu odbierania danych GPS. Dostęp jest kontrolowany za pomocą odpowiednich uprawnień w manifestie Androida aplikacji (plik `AndroidManifest.xml`). Dostępne są dwa uprawnienia:

ACCESS_COARSE_LOCATION - zapewnia aplikacji dostęp do dostawcy GPS oraz *ACCESS_FINE_LOCATION* - umożliwia aplikacji dostęp do sieci komórkowej i WiFi lokalizacji. Wymagane dla dostawcy sieci gdy *ACCESS_COARSE_LOCATION* jest nieustawiony.

2.5. Mapy Google

W zakładce Trening będzie wyświetlana również, korzystając z modułu GPS, aktualna pozycja użytkownika na Mapach Google. Dostęp do Map Google jest możliwy dzięki API (Maps SDK for Android) udostępnianego przez Google. Dostęp do tego API jest uzyskiwany z kluczem API, który został wygenerowany w panelu Google Cloud API. Możliwość wyświetlania pobranej mapy w aplikacji jest możliwa korzystając z pakietu NuGet Xamarin.Forms.Maps. Na mapie będzie także rysowana linią (klasa Polyline z wyżej wymienionego pakietu) trasa przebyta w trakcie treningu.

2.6. Baza danych SQLite

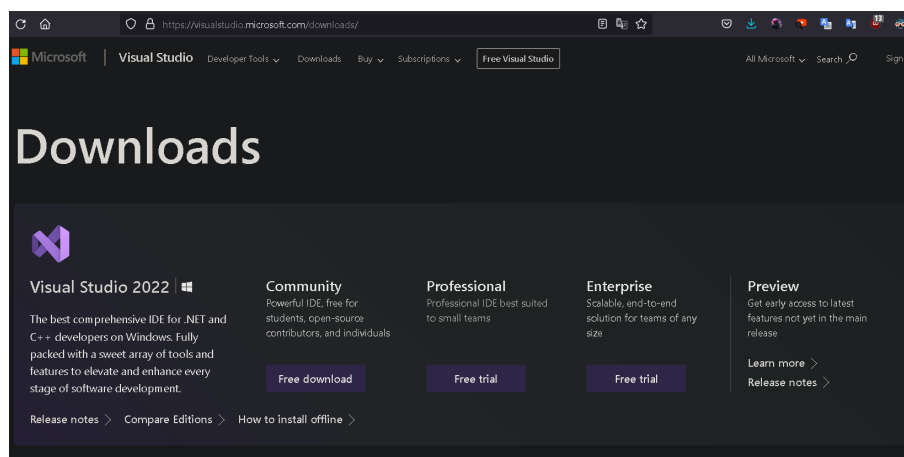
SQLite to mała, szybka i przystosowana do osadzania baza danych SQL oparta na systemie plików typu open source. Nie posiada oddzielnego komponentu serwowego, jak tradycyjne bazy danych.

3. Projektowanie

3.1. Przygotowanie narzędzi (Git, Visual Studio)

3.1.1. Visual Studio

Pierwszym i oczywistym krokiem, który musimy wykonać jest przygotowanie właściwych technologii i środowisk, które posłużą do tworzenia naszego projektu. Jako środowisko programistyczne wybrano Visual Studio 2022 i dostępną na nim wspomnianą wcześniej platformę Xamarin.Forms. W celu instalacji tego narzędzia, pobieramy plik instalacyjny odpowiedni dla naszego systemu ze strony Microsoftu¹, co pokazano na rysunku 3.1. Podczas instalacji, w menedżerze pakietów, wybieramy pakiety *Mobile i desktop development with ASP.NET*, (rys. 3.2), zaznaczając też opcjonalne dodatki w zależności od wymagań (rys. 3.3) (np. Xamarin, emulator Androida itd.). Środowisko Visual Studio posiada zintegrowany system kontroli wersji oprogramowania Git, który będzie wspomagał naszą pracę.

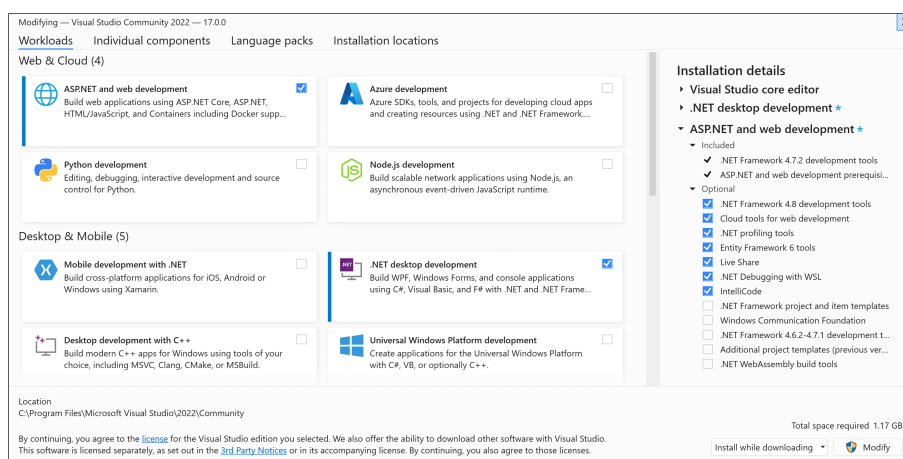


Rys. 3.1. Pobranie pliku instalacyjnego

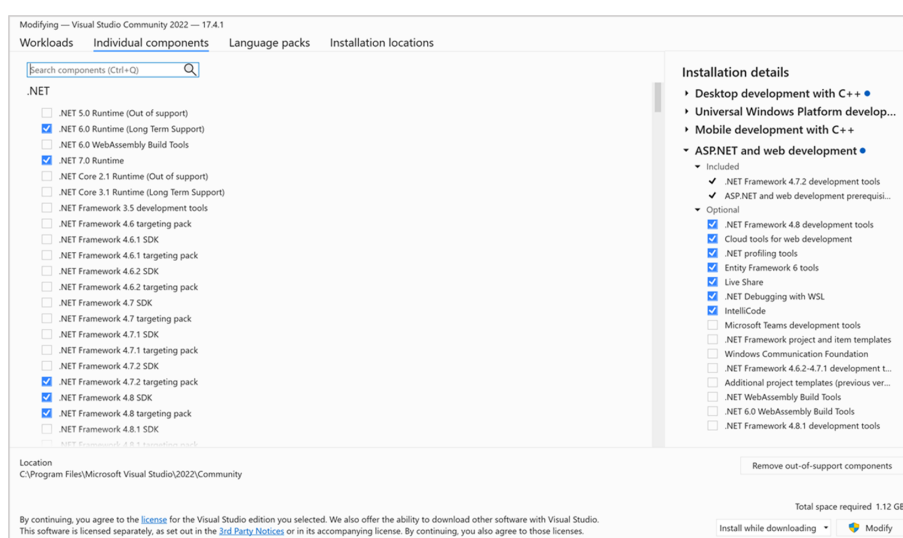
3.1.2. Git

Na stronie Github stworzone zostało zdalne repozytorium, do którego wysyłane będą kolejne wersje naszego projektu, z możliwością kontrybucji ze strony członków zespołu programistów. Lokalnie, na maszynach programistów zainstalowano dodatkowe narzędzie jakim jest desktopowa aplikacja Git for Windows, która dostarcza emulację BASH wykorzystywaną do uruchamiania Git'a z wiersza poleceń. Jest to duże udogodnienie dla osób, które czują się pewniej pracując z terminalem systemu

¹Plik instalacyjny na stronie [https://visualstudio.microsoft.com/pl/vs\[1\]](https://visualstudio.microsoft.com/pl/vs[1]).



Rys. 3.2. Wybór pakietów



Rys. 3.3. Dodatkowe pakiety i narzędzia

Linux. Plik instalacyjny można pobrać ze strony².

3.2. Szkice layoutów

3.2.1. Zakładka Kroki

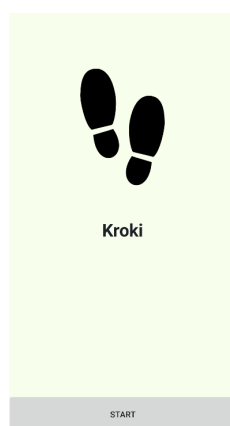
W zakładce Kroki rys. 3.4 (s. 11) użytkownik ma możliwość rozpoczęcia liczenia wykonanych kroków.

Liczenie kroków wykonuje się przy wykorzystaniu modułu akcelerometra..

3.2.2. Zakładka Trening

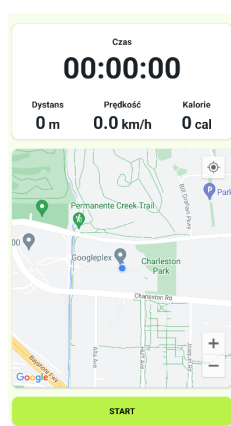
W zakładce Trening rys. 3.5 (s. 11) użytkownik ma możliwość rozpoczęcia tre-

²Plik instalacyjny na stronie <https://git-scm.com/download/win>[2].

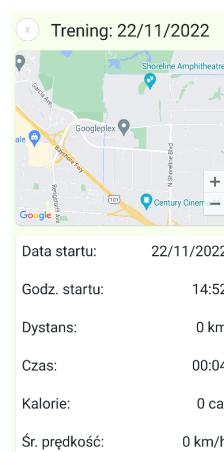


Rys. 3.4. Layout krokomierza

ningu. Zaczyna się odliczanie i wyświetlanie na ekranie bieżącego czasu odbywanego treningu, pokonany dystans w metrach, bieżąca prędkość oraz teoretyczna ilość spalonych kalorii. Na środkowej części ekranu wyświetlana zostaje, przy wykorzystaniu modułów GPS oraz Map Google bieżąca lokalizacja użytkownika. Za pomocą polilinii rysowana jest trasa, którą udaje się użytkownik.



Rys. 3.5. Layout treningu



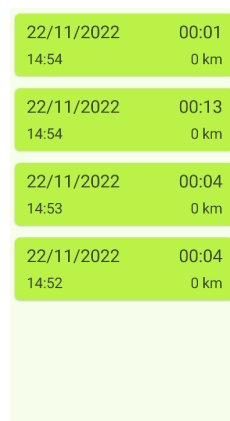
Rys. 3.6. Ekran podsumowania

Po zakończeniu treningu, w tej samej zakładce, wyświetlony zostaje ekran zawierający podsumowanie odbytego treningu. Jak pokazano na rys. 3.6 (s. 11), podsumowanie zawiera datę i czas rozpoczęcia treningu, przebyty dystans, czas treningu, spalone kalorie oraz średnią prędkość z jaką poruszał się użytkownik.

3.2.3. Zakładka Historia

W zakładce Historia, po zakończeniu kolejnych treningów, do bazy danych zapisywane są rekordy zawierające dane dotyczące odbytych w przeszłości treningów wraz z podstawowymi danymi jak data, czas oraz statystyki, co pokazano na rys.

3.7 (s. 12). Dane po zakończeniu treningu są najpierw zapisywane do bazy danych SQLite, następnie wyświetlane w zakładce Historia od najnowszego do najstarszego.



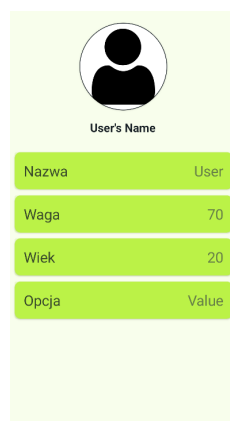
22/11/2022	00:01
14:54	0 km
22/11/2022	00:13
14:54	0 km
22/11/2022	00:04
14:53	0 km
22/11/2022	00:04
14:52	0 km


Rys. 3.7. Layout historii treningów

Wykorzystana w nim zostaje odpowiednia funkcja otwierająca widok statystyk i podsumowania.

3.2.4. Zakładka Ustawienia

W zakładce Ustawienia rys. 3.8 (s. 12) użytkownik ma możliwość konfiguracji własnego konta użytkownika, swoich danych fizycznych oraz wieku, na podstawie których obliczane zostają statystyki treningu jak np. spalone kalorie.



	
User's Name	
Nazwa	User
Waga	70
Wiek	20
Opcja	Value

Rys. 3.8. Layout ustawień i opcji

4. Implementacja

4.1. Wybrane kody z objaśnieniami działania

4.1.1. Zakładka Historia

Kod 1 przedstawia sposób inicjalizacji połączenia z bazą danych SQLite:

```
1 public History()  
2 {  
3     InitializeComponent();  
4  
5     _database = new SQLiteAsyncConnection(Path.Combine(Environment.  
6         GetFolderPath(Environment.SpecialFolder.LocalApplication  
7         Data), "trainingHistory.db3"));  
8 }
```

Listing 1. Połączenie z bazą danych

Kod 2 przedstawia sposób wypisania danych na ekran:

```
1 protected override async void OnAppearing()  
2 {  
3     base.OnAppearing();  
4  
5     //Wypisanie danych na ekran  
6     collectionView.ItemsSource = await GetTrainingData();  
7 }
```

Listing 2. Wypisanie danych na ekran

Kod 3 przedstawia sposób w jaki pobierane są dane z bazy danych:

```
1 public async Task<List<TrainingData>> GetTrainingData()  
2 {  
3     var query = await _database.Table<TrainingData>().ToListAsync()  
4     ;  
5     results = Enumerable.Reverse(query).ToList();  
6     return results;  
7 }
```

Listing 3. Pobranie danych z bazy

Kod 4 zawiera funkcję otwierającą widok statystyk i podsumowania:

```
1 private async void OpenStatistics(object sender, EventArgs e)  
2 {  
3     var btn = (Button)sender;  
4     await Navigation.PushAsync(new StatisticsView(btn.ClassId));  
5 }
```

```
5 }
```

Listing 4. Otwieranie widoku statystyk i podsumowania

Kod 5 przedstawia sposób w jaki dane z bazy danych zostaną wypisane na ekran:

```
1 public async Task<List<TrainingData>> GetTrainingData()
2 {
3     var query = _database.Table<TrainingData>().Where(p => p.Id ==
4     buttonId);
5     var result = await query.ToListAsync();
6     nazwa.Text = "Trening: " + result[0].DateDay;
7
8     _databaseTraining = new SQLiteAsyncConnection(Path.Combine(
9     Environment.GetFolderPath(Environment.SpecialFolder.
10    LocalApplicationData), result[0].TrainingDatabase));
11     var locations = await _databaseTraining.Table<CurrentData>().
12     ToListAsync();
13
14     Polyline polyline = new Polyline();
15     polyline.StrokeColor = Color.FromHex("#192126");
16     polyline.StrokeWidth = 20;
17     foreach (var pos in locations)
18     polyline.Geopath.Add(new Position(pos.Latitude, pos.Longitude))
19     ;
20     map.MapElements.Add(polyline);
21
22     Position startPos = new Position(locations[0].Latitude,
23     locations[0].Longitude);
24     map.MoveToRegion(MapSpan.FromCenterAndRadius(startPos, Distance
25     .FromKilometers(1)));
26
27     dataStartu.Text = result[0].DateDay;
28     godzinaStartu.Text = result[0].DateTime;
29     dystans.Text = result[0].Distance;
30     czas.Text = result[0].Time;
31     calories.Text = result[0].Calories.ToString() + " cal";
32     avrSpeed.Text = result[0].AvrSpeed.ToString() + " km/h";
33     return result;
34 }
```

Listing 5. Wypisanie danych z bazy danych na ekran

4.1.2. Zakładka Trening

Kod 6 przedstawia sposób inicjalizacji połączenia z bazą danych:

```
1 public Training()
2 {
3     InitializeComponent();
4
5     _database = new SQLiteAsyncConnection(Path.Combine(Environment.
6         GetFolderPath(Environment.SpecialFolder.
7         LocalApplicationData), "trainingHistory.db3"));
8     _database.CreateTableAsync<TrainingData>();
9     GetLocation();
10 }
```

Listing 6. Połączenie z bazą danych

Kod 7 zawiera funkcję pobierającą lokalizację:

```
1 private async void GetLocation()
2 {
3     //pobranie lokalizacji
4     var request = new GeolocationRequest(GeolocationAccuracy.Best);
5     var location = await Geolocation.GetLocationAsync(request);
6
7     if (location != null)
8     {
9         position = new Position(location.Latitude, location.Longitude
10     );
11
12     //przesunięcie widoku na lokalizację przeciwnika
13     map.MoveToRegion(MapSpan.FromCenterAndRadius(position,
14         Distance.FromKilometers(0.5)));
15
16     if (isTraining)
17     {
18         //zapis danych lokalizacji
19         positionsList.Add(new PositionList { Location = location,
20             TimeLasted =
21             (hours * 3600 + mins * 60 + secs) });
22         await SaveCurrentData(new CurrentData
23         {
24             Latitude = location.Latitude,
25             Longitude = location.Longitude,
26             TimeLasted = (hours * 3600 + mins * 60 + secs)
27         });
28     }
```

```

26         UpdateInfo();
27     }
28
29 }
30
31 //pobranie lokalizacji wywoływane co 2 sekundy
32 await Task.Delay(2000);
33 GetLocation();
34
35 }

```

Listing 7. Pobranie lokalizacji

Kod 8 wprowadza możliwość rozpoczęcia treningu:

```

1 private void StartTraining(object sender, EventArgs e)
2 {
3     btnStartF.IsVisible = false;
4     btnResumeF.IsVisible = false;
5     btnEndF.IsVisible = false;
6     btnStopF.IsVisible = true;
7     isTraining = true;
8     startDate = DateTime.Now;
9
10    //baza danych lokalizacji w treningu
11    _databaseTraining = new SQLiteAsyncConnection(Path.Combine(
    Environment.GetFolderPath(Environment.SpecialFolder.
    LocalApplicationData), "training" + startDate.ToString("
    dd-MM-yyyy-HH-mm-ss") + ".db3"));
12    _databaseTraining.CreateTableAsync<CurrentData>();
13
14    //inicjalizacja timera
15    timer = new Timer();
16    timer.Interval = 1000;
17    timer.Elapsed += Timer_Elapsed;
18    timer.Start();
19 }

```

Listing 8. Rozpoczęcie treningu

Kod 9 zawiera funkcję liczącą sekundy, wywoływaną co sekundę:

```

1 private void Timer_Elapsed(object sender, ElapsedEventArgs e)
2 {
3     secs++;
4     if (secs == 59)
5     {
6         mins++;

```



```

7     secs = 0;
8 }
9 if (mins == 59)
10 {
11     hours++;
12     mins = 0;
13 }
14 Device.BeginInvokeOnMainThread(() =>
15 {
16     timerValue.Text = string.Format("{0:00}:{1:00}:{2:00}", hours
17     , mins, secs);
18 });
19 }

```

Listing 9. Liczenie sekund (co sekundę)

Kod 10 umożliwia zakończenie treningu oraz obsługę danych po odbytym treningu:

```

1 private async void EndTraining(object sender, EventArgs e)
2 {
3     btnResumeF.IsVisible = false;
4     btnEndF.IsVisible = false;
5     btnStopF.IsVisible = false;
6     btnStartF.IsVisible = true;
7
8     //zapis danych treningu do glownej bazy danych
9     await SaveTrainingData(new TrainingData
10     {
11         DateDay = startDate.ToString("dd/MM/yyyy"),
12         DateTime = startDate.ToString("HH:mm"),
13         Time = string.Format("{0:00}:{1:00}", (hours * 60 + mins),
14         secs),
15         Distance = way.ToString() + " km",
16         Calories = caloriesBurned,
17         AvrSpeed = avrSpeed,
18         TrainingDatabase = "training" + startDate.ToString("
19         dd_MM_yyyy_HH_mm_ss") + ".db3"
20     });
21
22     //otwarcie strony podsumowania treningu
23     await Navigation.PushAsync(new StatisticsView(CountTrainings().
24     Result.ToString()));
25
26     //wyzerowanie zmiennych pomiarowych
27     way = 0;
28 }

```

```

25     tempo = 0;
26     caloriesBurned = 0;
27     avrSpeed = 0;
28     speedSum = 0;
29     timerValue.Text = "00:00:00";
30     amountDistance.Text = "0";
31     amountCalories.Text = "0";
32     amountSpeed.Text = "0.0";
33     hours = 0;
34     mins = 0;
35     secs = 0;
36     map.MapElements.Clear();
37 }

```

Listing 10. Zakończenie treningu zapis danych do bazy i ich podsumowanie

Kod 11 zawiera funkcję aktualizującą informacje na ekranie:

```

1  private void UpdateInfo()
2  {
3      //polyline rysuje trasę na mapie
4      Polyline polyline = new Polyline();
5      polyline.StrokeColor = Color.FromHex("#192126");
6      polyline.StrokeWidth = 20;
7      int length = positionsList.Count - 1;
8      double tmpWay;
9      if (length > 1)
10     {
11         //ustawienie lini na dwie ostatnie pozycje
12         polyline.Geopath.Add(new Position(positionsList[length - 1].
13             Location.Latitude, positionsList[length - 1].Location.Longitude)
14             );
15         polyline.Geopath.Add(new Position(positionsList[length].
16             Location.Latitude, positionsList[length].Location.Longitude));
17         //wywołanie linii na mapie
18         map.MapElements.Add(polyline);
19
20         //obliczenia parametrow treningu
21         tmpWay = Location.CalculateDistance(positionsList[length].
22             Location, positionsList[length - 1].Location, DistanceUnits.
23             Kilometers);
24         way = Math.Round((way + tmpWay), 3);
25         tempo = (tmpWay / (positionsList[length].TimeLasted -
26             positionsList[length - 1].TimeLasted)) * 3600;
27         tempo = Math.Round(tempo, 1);
28         speedSum += tempo * (positionsList[length].TimeLasted -

```

```
positionsList[length - 1].TimeLasted);
23     avrSpeed = speedSum / (positionsList[length].TimeLasted);
24     avrSpeed = Math.Round(avrSpeed, 1);
25     //kalorie na minute = (MET * 3.5 * waga) / 200
26     //MET - ile razy więcej kalorii spala przy danej aktywnosci w
    porownaniu do odpoczynku
27     caloriesBurned = ((avrSpeed * 3.5 * weight) / 200) * (((
double)positionsList[length].TimeLasted) / 60);
28     caloriesBurned = Math.Round(caloriesBurned, 1);
29 }
30
31 //wyswietlenie dystansu w okreslonym formacie zaleznym od
dystansu
32 if (way < 1)
33 {
34     amountDistance.Text = (way * 1000).ToString();
35     distanceSize.Text = " m";
36 }
37 else if (way < 10)
38 {
39     amountDistance.Text = string.Format("{0:0.00}", way);
40     distanceSize.Text = " km";
41 }
42 else if (way < 100)
43 {
44     amountDistance.Text = string.Format("{0:00.0}", way);
45     distanceSize.Text = " km";
46 }
47 else
48 {
49     amountDistance.Text = string.Format("{0:000}", way);
50     distanceSize.Text = " km";
51 }
52
53 //wyswietlenie predkosci w okreslonym formacie
54 if (tempo < 10)
55     amountSpeed.Text = string.Format("{0:0.0}", tempo);
56 else
57     amountSpeed.Text = string.Format("{0:00}", tempo);
58 //wyswietlenie spalonych kalorii
59 if (caloriesBurned < 10)
60     amountCalories.Text = caloriesBurned.ToString();
61 else
62     amountCalories.Text = string.Format("{0:00}", caloriesBurned);
```

63 }

Listing 11. Aktualizowanie informacji na ekranie

Kod 12 wykonuje zapis danych treningu do bazy danych:

```

1 public Task<int> SaveTrainingData(TrainingData trainingData)
2 {
3     return _database.InsertAsync(trainingData);
4 }

```

Listing 12. Zapis danych do bazy danych**4.1.3. Zakładka Kroki**

Kod 13 implementuje modułu Step_Counter, wykorzystujący czujnik akcelerometr:

```

1 public partial class StepCounter : ContentPage
2 {
3     //inicjalizacja listy przechowującej liczbę kroków
4     List<double> accData = new List<double>();
5     int stepsNumber = 0;
6     DateTime czas = DateTime.Now;
7     TimeSpan interval = TimeSpan.FromSeconds(0.1);
8
9
10    public StepCounter()
11    {
12        InitializeComponent();
13
14        Accelerometer.ReadingChanged += Accelerometer_ReadingChanged;
15    }
16
17    //implementacja modułu akcelerometra
18    void Accelerometer_ReadingChanged(object sender,
19    AccelerometerChangedEventArgs args)
20    {
21        if (DateTime.Now - czas > interval)
22        {
23            czas = DateTime.Now;
24            var xVal = args.Reading.Acceleration.X * 10;
25            var yVal = args.Reading.Acceleration.Y * 10;
26            var zVal = args.Reading.Acceleration.Z * 10;
27            var accValue = Math.Sqrt(xVal * xVal + yVal * yVal + zVal *
28            zVal) - 10;
29            if (accData.Count > 240)
30                accData.RemoveAt(0);
31        }
32    }
33 }

```

```

29     accData.Add(accValue);
30
31     //petla liczaca kroki
32     if (accData.Count > 1)
33     {
34         for (int i = 1; i < accData.Count - 1; i++)
35         {
36             if (accData[i] > 1)
37             {
38                 if (accData[i] > accData[i - 1] && accData[i] >
accData[i + 1])
39                 {
40                     stepsNumber++;
41                     accData.Clear();
42                 }
43             }
44
45         }
46     }
47     amountSteps.Text = stepsNumber.ToString();
48 }
49 }

```

Listing 13. Krakomierz i akcelerometr

Kod 14 zawiera funkcję dającą możliwość rozpoczęcia i zakończenia liczenia kroków:

```

1  void Button_Clicked(object sender, EventArgs e)
2  {
3      try
4      {
5          if (Accelerometer.IsMonitoring)
6          {
7              Accelerometer.Stop();
8              btn.Text = "Start";
9          }
10         else
11         {
12             Accelerometer.Start(SensorSpeed.UI);
13             btn.Text = "Stop";
14         }
15     }
16 }
17 catch (FeatureNotSupportedException fnsEx)
18 {

```

```
19     // Not supported on device
20 }
21 catch (Exception ex)
22 {
23     // Something else went wrong
24 }
25 }
```

Listing 14. Rozpoczęcia i zakończenie liczenia kroków

5. Testowanie

6. Podręcznik użytkownika

Bibliografia

- [1] *Strona internetowa firmy SELS*. URL: <http://www.sels.com.pl/index.php?cPath=1> (term. wiz. 29.10.2012).
- [2] *Strona internetowa TexLive*. URL: <https://www.tug.org/texlive/acquire-netinstall.html> (term. wiz. 08.10.2022).

Spis rysunków

1.1. Ekran treningu	5
1.2. Ekran podsumowania	5
1.3. Ekran krokomierza	5
3.1. Pobranie pliku instalacyjnego	9
3.2. Wybór pakietów	10
3.3. Dodatkowe pakiety i narzędzia	10
3.4. Layout krokomierza	11
3.5. Layout treningu	11
3.6. Ekran podsumowania	11
3.7. Layout historii treningów	12
3.8. Layout ustawień i opcji	12

Spis tabel

Spis listingów

1.	Połączenie z bazą danych	13
2.	Wypisanie danych na ekran	13
3.	Pobranie danych z bazy	13
4.	Otwieranie widoku statystyk i podsumowania	13
5.	Wypisanie danych z bazy danych na ekran	14
6.	Połączenie z bazą danych	15
7.	Pobranie lokalizacji	15
8.	Rozpoczęcie treningu	16
9.	Liczenie sekund (co sekundę:)	16
10.	Zakończenie treningu zapis danych do bazy i ich podsumowanie . . .	17
11.	Aktualizowanie informacji na ekranie	18
12.	Zapis danych do bazy danych	20
13.	Krakomierz i akcelerometr	20
14.	Rozpoczęcia i zakończenie liczenia kroków	21