PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA
W NOWYM SĄCZU

Instytut Techniczny
Informatyka Stosowana

DOKUMENTACJA PROJEKTOWA

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Aplikacja śledząca trasy rowerowe

Autorzy:

Jan Wilczyński
Arkadiusz Rajski

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2021

Spis treści

1. Ogólne określenie wymagań

1.1. Zamówienie aplikacji przez klienta

Prowadzę sklep rowerowy i coraz więcej klientów narzeka na ograniczenia aplikacji "Strava". Brakuje im m.in. szczegółowego zapisywania tras, porównywania ich oraz śledzenia postępu. Nie ma też możliwości dodawania zdjęć z aktywności fizycznej. Zdecydowałem, że zamówię u Państwa tego typu aplikację, której wymagania przedstawiam poniżej:

1) Każdy człowiek posługujący się rowerem powinien uznać ją za przydatną, dlatego muszą się w niej znajdować zarówno elementy dla osób dojeżdżających tym pojazdem do pracy, jak i dla turystów przemierzających długie dystanse, czy wreszcie sportowców.

2) Szata graficzna powinna być jednolita i nowoczesna. Powinna składać się z jednego dominującego koloru i jego odcieni dla poszczególnych przycisków i paneli. Wymaganie to nie dotyczy tekstu - może mieć różne kolory, ważne jest aby był dobrze widoczny i spełniał rozmaite zadania, takie jak dobrze widoczne nagłówki czy delikatnie nakreślone podpowiedzi.

3) Chciałbym aby logo było umieszczone w lewym górnym rogu, a dostęp do poszczególnych funkcji odbywał się poprzez menu zakładkowe umieszczone z lewej strony ekranu.

W menu powinny znaleźć się poszczególne zakładki:

- Trasa (Możliwość rozpoczęcia śledzenia trasy, rekomendacje tras pod względem poziomu trudności i atrakcyjności dla turystów).

- Historia tras (Utrzymywanie historii przebytych tras, możliwość dodania ulubionej trasy, oraz planowanie tras "do wykonania". W dniu w którym trasa ma zostać wykonana aplikacja powinna przypomnieć o tym użytkownikowi poprzez powiadomienie.

- Pomiary (Mierzenie aktualnej prędkości jazdy, ostrości zakrętów i wysokości nad poziomem morza oraz gromadzenie tych danych. Powinny one być pokazywane od 10 minut do 6 godzin wstecz, w zależności od konfiguracji użytkownika. Sekcja powinna wskazywać momenty w których rower się zatrzymywał).

- Statystyki (Przechowują dane zebrane poprzez pomiary, zbieranie statystyk można włączyć i wyłączyć za pomocą kontrolki w ustawieniach aplikacji. Statystyki powinny automatycznie zbierać takie dane jak prędkość średnia, prędkość maksymalna, średnie przewyższenie tras. Użytkownik może przeglądać dane za pomocą wykresów oraz dobrać ramy czasowe (np. 20 października do 10 listopada). Statystyki powinny być dostępne bezterminowo, należy jednak pozwolić użytkownikowi na ich usuwanie).

- Ustawienia (Konfiguracja różnych ustawień aplikacji: sposób wyświetlania historii tras, statystyk, motywu, sposobu wyświetlania poszczególnych danych jak np. pomiary czy statystyki, ustawienia częstotliwości powiadomień. W przypadku dużej ilości ustawień po kliknięciu przycisku "Ustawienia"użytkownik najpierw powinien zobaczyć listę kategorii, a dopiero potem konkretne ustawienia).

4) Przy przełączaniu między zakładkami albo ładowaniu ekranów powinna wyświetlać się animowana ikonka ładowania (loader) z informacją typu "Proszę czekać", "Trwa ładowanie" itp.

5) Aplikacja powinna przystosować swoją szatę graficzną do ilości światła oraz pory dnia.

6) Szata graficzna aplikacji będzie utrzymywana w dwóch kolorach: ciemnym (tło w ciemnym odcieniu, ikony oraz grafika w jasnym, tekst w kolorze białym) i jasnym (tło w jasnym odcieniu, ikony oraz grafika w ciemnym, tekst w kolorze czarnym). Kolor motywu będzie dostępny do wyboru przez użytkownika.

7) Ekran powinien być podzielony na dwie części: wcześniej wspomniane boczne menu oraz część główną. Menu powinno znajdować się z lewej strony oraz być podzielone w pionie na równej wielkości przyciski, z których najwyżej ustawiony powinien zawierać logo aplikacji. Aktywny przycisk będzie w widoczny sposób połączony z tłem części głównej.

8) Po rozpoczęciu trasy aplikacja powinna udostępniać użytkownikowi mapę wraz z trasą oraz śledzeniem jego lokalizacji. Użytkownik może również poruszać się bez wyznaczonej trasy. Dodatkowo na górze powinien znajdować się kompas wskazujący aktualny kierunek.

9) GPS powinien mierzyć dane z dokładnością co do metra.

10) Aplikacja powinna powiadomić użytkownika w przypadku kończącej się pamięci urządzenia.

2. Określenie wymagań szczegółowych

2.1. Odpowiedź na zamówienie klienta

Szanowny Panie,
Możemy od razu zacząć realizować zamówienie, ale najpierw chciałbym omówić szczegóły techniczne.

Odniosę się do przesłanych przez Pana punktów i przedstawię rzeczy wymagające poprawy/zmiany.

1) W punkcie trzecim wspomina Pan o menu i przycisku "Trasa". Jeśli chodzi o poziom trudności trasy, to trzeba ustalić na jakiej podstawie ma być on wyznaczany. Moją propozycją jest tutaj stosunek przewyższenia do każdego kilometra trasy. Co do poziomu atrakcyjności dla turystów, niestety nie jest to bezpośrednio możliwe, ponieważ musielibyśmy stworzyć serwer oraz zbierać opinie i oceny od użytkowników, ale implementując Mapy Google użytkownik będzie widział ocenę miejsca którego szuka, co jest poniekąd spełnieniem Pana wymogu.

- Jeśli chodzi o przycisk "Historia tras" i planowanie trasy, to zamiast planowania gotowej trasy (bo przecież miejsce startu użytkownika może być różne z wielu powodów) proponuję dodać planowanie samego celu trasy. Wtedy użytkownik dostanie powiadomienie o planowanym celu i będzie mógł bezpośrednio przejść do nawigacji do określonej lokalizacji.

- Mierzenie ostrości zakrętów można poniekąd osiągnąć za pomocą akcelerometra. Wymagałoby to zbierania danych pokonując łatwe, średnie i trudne zakręty, ale będzie to trudne do uzyskania a dane i tak mogą być bardzo niedokładne. Zamiast tego proponuje dodać pomiar przeciążenia na całej trasie, co rzuci światło na oczekiwaną przez Pana "ostrość" trasy.

- Jeśli chodzi o pokazywanie pomiarów od 10 minut do 6 godzin wstecz, to proponuję dodać ustawienie do ilu minut/godzin wstecz pomiary mają być pokazywane, oraz przedstawiać je w formie wykresu.

2) W punkcie czwartym wspomina Pan o ekranie ładowania, który w mojej opinii nie jest potrzebny, ponieważ nie będzie interakcji ze zdalną bazą danych. Dane będą pobierane bezpośrednio z urządzenia na którym zainstalowana jest aplikacja, co

będzie działo się błyskawicznie. Nie ma więc potrzeby dodawania ekranu ładowania, który będzie się wyświetlał setne części sekundy.

3) W punkcie szóstym proponuję odwrócić kolory tekstu. W jasnym motywie czarny tekst będzie kontrastowy i widoczny, a w ciemnym lepszym kolorem tekstu będzie wyróżniający się biały. Poprawi to czytelność aplikacji.

4) Jeśli chodzi o punkt dziewiąty, to niestety nie jest to możliwe do wykonania. Dokładność modułów GPS w telefonach oscyluje w granicach 5 metrów zarówno jeśli chodzi o położenie horyzontalne, jak i o wysokość urządzenia.

Pozostałe punkty na ten moment leżą w zakresie naszych możliwości. Do realizacji projektu możemy przystąpić bezzwłocznie.

Do odpowiedzi załączam zrzut ekranu zawierający proponowany wygląd ekranu głównego aplikacji, widoczny jako rysunek 2.1.

Rys. 2.1. Propozycja ekranu głównego

3. Projektowanie

3.1. Narzędzia

Do wykonania aplikacji użyte zostaną następujące narzędzia:

- Visual Studio 2019

- Xamarin Forms v5.0.0.2125

- GitHub

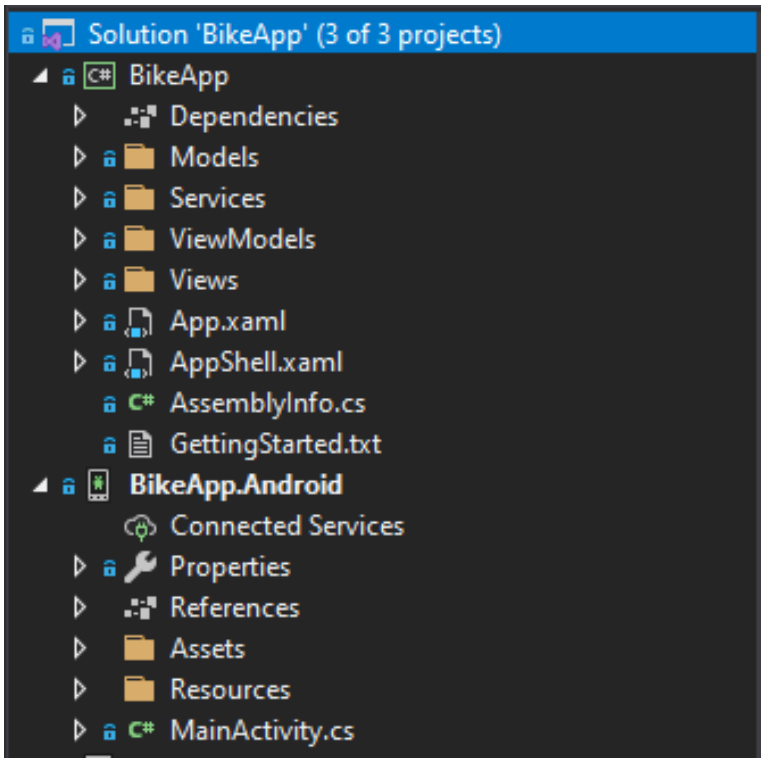
- Nuget packages

- Android Emulator(Pixel 2 Pie 9.0)

Rys. 3.1. Szkic strony głównej aplikacji

3.2. Xamarin.Forms

Xamarin Forms to framework pozwalający pisać aplikacje na systemy takie jak m.in. Android oraz iOS. Łączy on głównie język C# oraz XML. Polega na budowaniu struktury komponentów składającej się z widoków, modeli widoków oraz zawartości (view(page), viewmodel, content) które służą do projektowania wyglądu oraz sposobu działania aplikacji.
Struktura projektu (rozwiązania, wg. nazewnictwa wykorzytanego w projektowaniu środowiska Visual Studio) została zaprezentowana na rysunku 3.2.



Rys. 3.2. Struktura projektu widoczna w programie Visual Studio

4. Implementacja

4.1. Pierwsze kroki

Po stworzeniu projektu poznawaliśmy strukturę oraz działanie aplikacji próbując modyfikować istniejące już klasy. Po zapoznaniu się z podstawami realizowaliśmy dalsze tutoriale aby swobodnie pracować w środowisku Xamarin'a.

4.2. Modyfikacja menu

Pierwszym zadaniem było ustalenie szaty graficznej, widoku podstawowego z przyciskiem oraz menu. Aby to zrobić należało wykonać następujące czynności:

-Dopisać odpowiednie właściwości do kodu XML, takie jak background color, fontsize, padding, textcolor itp., aby wystylizować układ widoku. Na rysunku 4.1 znajduje się fragment wprowadzonych modyfikacji.

```
      <Shell.Resources>
          <ResourceDictionary>
              <Style x:Key="BaseStyle" TargetType="Element">
-                 <Setter Property="Shell.BackgroundColor" Value="{StaticResource Primary}" />
+                 <Setter Property="Shell.BackgroundColor" Value="#181818" />
                  <Setter Property="Shell.ForegroundColor" Value="White" />
                  <Setter Property="Shell.TitleColor" Value="White" />
                  <Setter Property="Shell.DisabledColor" Value="#B4FFFFFF" />
@@ -40,13 +41,13 @@
                          <VisualStateGroup x:Name="CommonStates">
                              <VisualState x:Name="Normal">
                                  <VisualState.Setters>
-                                     <Setter Property="BackgroundColor" Value="{x:OnPlatform UWP=Transparent, iOS=White}" />
+                                     <Setter Property="BackgroundColor" Value="#181818" />
                                      <Setter TargetName="FlyoutItemLabel" Property="Label.TextColor" Value="{StaticResource Primary}" />
                                  </VisualState.Setters>
                              </VisualState>
                              <VisualState x:Name="Selected">
                                  <VisualState.Setters>
-                                     <Setter Property="BackgroundColor" Value="{StaticResource Primary}" />
+                                     <Setter Property="BackgroundColor" Value="#252525" />
                                  </VisualState.Setters>
                              </VisualState>
                          </VisualStateGroup>
@@ -79,7 +80,7 @@
          https://docs.microsoft.com/dotnet/api/xamarin.forms.shellgroupitem.flyoutdisplayoptions?view=xamarin-forms
      -->
      <!--Menu items-->
-     <FlyoutItem Title="BikeApp" Icon="icon_about.png">
+     <FlyoutItem Title="BikeApp" Icon="app_logo_raw.png">
          <ShellContent Route="AboutPage" ContentTemplate="{DataTemplate local:AboutPage}" />
      </FlyoutItem>
      <FlyoutItem Title="Browse" Icon="icon_feed.png">
```

Rys. 4.1. Dodanie właściwości stylujących widok

-Stworzyć widok główny na podstawie języka XML oraz metody w języku C# informującej o kliknięciu w przycisk (zgodnie z rysunkiem 4.2).

```
<StackLayout BackgroundColor ="#292929" Orientation="Vertical" Padding="30,24,30,24" Spacing="10" Margin="0,0,0,0">
    <Label Text="Start tracking your activity" FontSize="Title" TextColor="White" HorizontalTextAlignment="Center"/>
    <Label Text="Track your activity, compare your riding statistics, make photo-history of your favourite trips." FontSize="16" Padding="0,0,0,0" TextCo
    <Label Text="Press to start tracking" FontSize="16" Padding="0,0,0,0" TextColor="White" HorizontalTextAlignment="Center"/>

    <Button Margin="0,10,0,0" Text="Start tracking"
            Clicked="Button_Clicked"
            BackgroundColor="#4934eb"
            TextColor="White" />
</StackLayout>
```

Rys. 4.2. Implementacja widoku

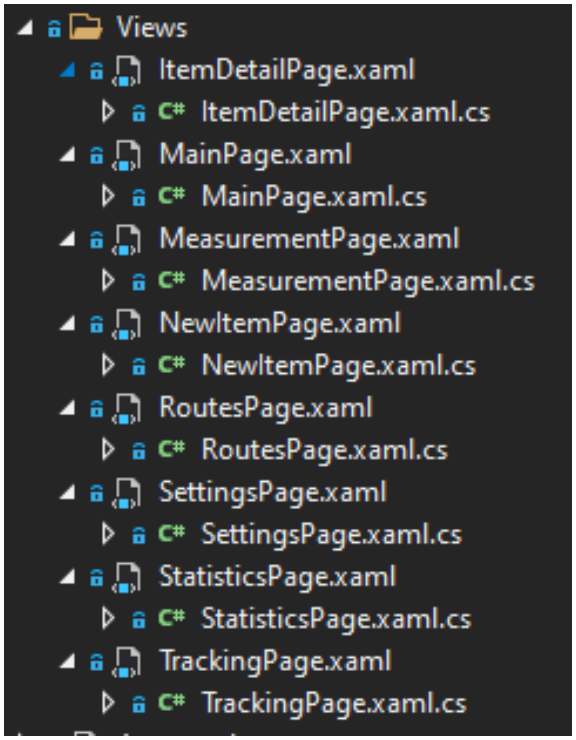
-Pobrać i zainstalować paczkę 'acr' z NuGet Packages, zaimplementować opcję alertu, zaktualizować wersję Xamarin.forms z 5.0.0.2012 do 5.0.0.2125 z powodu konfliktu biblioteki emulującej system Aandroid w języku Java (rysunek 4.3).

```
135         -     <PackageReference Include="Xamarin.Forms" Version="5.0.0.2012" />
136   135         <PackageReference Include="Xamarin.Essentials" Version="1.6.1" />
      136   +     <PackageReference Include="Xamarin.Forms">
      137   +       <Version>5.0.0.2125</Version>
      138   +     </PackageReference>
```

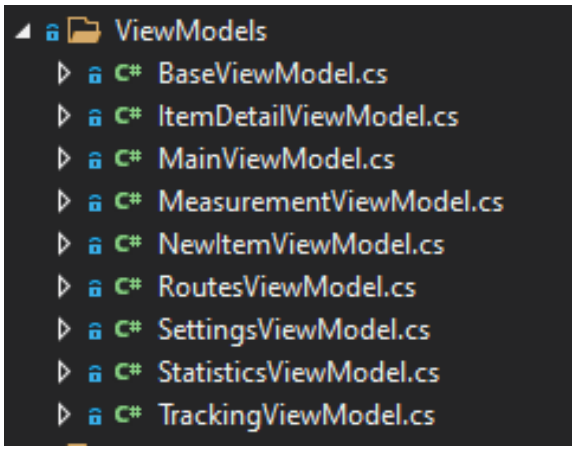
Rys. 4.3. Modyfikacja NuGet Packages

4.3. Dodanie oraz wystylizowanie elementów menu

Na początku należało utworzyć odpowiednie widoki aby stworzyć spójny szkielet aplikacji oraz stworzyć odpowiadające im modele widoków (rysunki 4.4 oraz 4.5).



Rys. 4.4. Szkielet widoków



Rys. 4.5. Szkielet modeli widoków

Następnym krokiem było ustawienie odpowiednich parametrów w celu połączenia poszczególnych widoków z ich modelami (zastosowano w tym celu dyrektywę Binding). Na rysunku 4.6 zaprezentowano jedną z takich modyfikacji.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="BikeApp.Views.MeasurementPage"
             BackgroundColor="{StaticResource Background}"
             xmlns:vm="clr-namespace:BikeApp.ViewModels"
             Title="{Binding Title}">

    <ContentPage.BindingContext>
        <vm:SettingsViewModel />
    </ContentPage.BindingContext>
```

Rys. 4.6. Łączenie ModelView z View

Kolejnym krokiem było ustawienie odpowiednich opcji FlyoutMenu (rysunek 4.7).

```
<FlyoutItem Title="Start tracking" Icon="tracking_icon.png">
    <ShellContent Route="TrackingPage" ContentTemplate="{DataTemplate local:TrackingPage}" />
</FlyoutItem>

<FlyoutItem Title="Your routes" Icon="route_icon.png">
    <ShellContent Route="RoutesPage" ContentTemplate="{DataTemplate local:RoutesPage}" />
</FlyoutItem>

<FlyoutItem Title="Live measurement" Icon="measurement_icon.png">
    <ShellContent Route="MeasurementPage" ContentTemplate="{DataTemplate local:MeasurementPage}" />
</FlyoutItem>

<FlyoutItem Title="Statistics" Icon="statistics_icon.png">
    <ShellContent Route="StatisticsPage" ContentTemplate="{DataTemplate local:StatisticsPage}" />
</FlyoutItem>

<FlyoutItem Title="Settings" Icon="settings_icon.png">
    <ShellContent Route="SettingsPage" ContentTemplate="{DataTemplate local:SettingsPage}" />
</FlyoutItem>
```

Rys. 4.7. Dodanie opcji menu

Po zakończeniu zostały stworzone oraz dodane odpowiednie ikony wraz z tytułami do każdego widoku (rysunki od 4.8 do 4.10)

```
<ItemGroup>
  <AndroidResource Include="Resources\drawable\settings_icon.png" />
</ItemGroup>
```

Rys. 4.8. Dodanie ikon

```
+ namespace BikeApp.ViewModels
+ {
+     public class MainViewModel : BaseViewModel
+     {
+         public MainViewModel()
+         {
+             //Name of current tab
+             Title = "Main page";
+         }
+     }
+ }
```

Rys. 4.9. Dodanie tytułów widoków

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="BikeApp.Views.TrackingPage"
             xmlns:vm="clr-namespace:BikeApp.ViewModels"
             Title="{Binding Title}">
```

Rys. 4.10. Dodanie styli

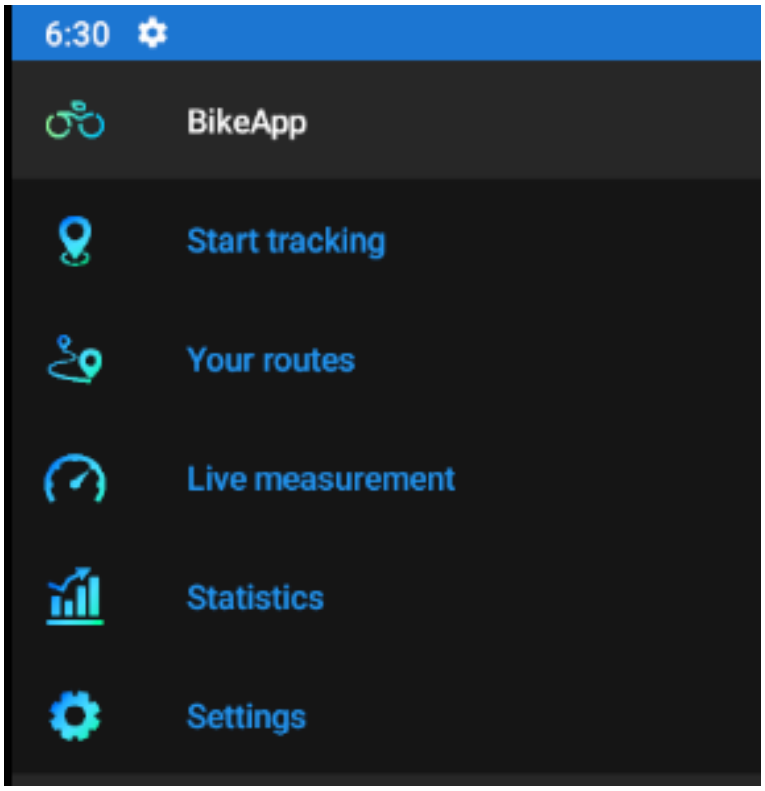
Ostatnim etapem, przedstawionym na rysunku 4.11, było wystylizowanie oraz poprawa ewentualnych błędów w kodzie.

```
<Grid>
    <ScrollView>
        <StackLayout BackgroundColor ="#292929"
                     Orientation="Vertical"
                     Padding="30,24,30,24"
                     Spacing="10"
                     Margin="0,0,0,0">
            <Label Text="Start tracking your activity"
                   FontSize="Title"
                   TextColor="White"
                   HorizontalTextAlignment="Center"/>
            <Label Text="Track your activity, compare your riding statistics, make photo-history of your favourite trips."
                   FontSize="16"
                   Padding="0,0,0,0"
                   TextColor="White"
                   HorizontalTextAlignment="Center"/>
            <Label Text="Press to start tracking"
                   FontSize="16" Padding="0,0,0,0"
                   TextColor="White"
                   HorizontalTextAlignment="Center"/>

            <Button Margin="0,10,0,0" Text="Start tracking"
                    Clicked="Button_Clicked"
                    BackgroundColor="#4934eb"
                    TextColor="White" />
        </StackLayout>
    </ScrollView>
</Grid>
```

Rys. 4.11. Finalizacja

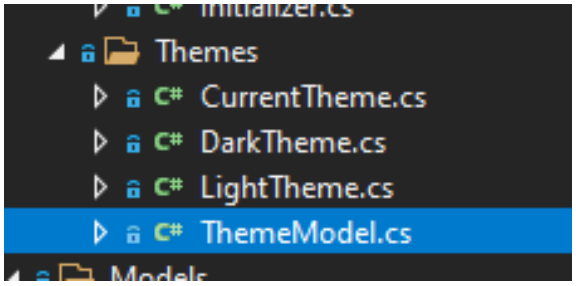
Efekt końcowy został zaprezentowany na rysunku 4.12.



Rys. 4.12. Wygląd menu po ukończeniu prac

4.4. Dodanie motywu jasnego i ciemnego

Praca została rozpoczęta od utworzenia klas reprezentujących motywy i przechowujących odpowiadające im palety kolorów aplikacji (rysunki od 4.13 do 4.16).



Rys. 4.13. Układ klas motywów

```
3 references
public abstract class ThemeModel
{
    3 references
    public string BackgroundColor1 { get; set; }
    3 references
    public string BackgroundColor2 { get; set; }
    3 references
    public string BackgroundColor3 { get; set; }
    3 references
    public string ButtonColor { get; set; }
    3 references
    public string TextColor { get; set; }
}
```

Rys. 4.14. Kod abstrakcyjnej klasy bazowej

```
public class DarkTheme : ThemeModel
{
    – references
    public DarkTheme()
    {
        BackgroundColor1 = "#222222";
        BackgroundColor2 = "#333333";
        BackgroundColor3 = "#444444";
        ButtonColor = "#4934eb";
        TextColor = "#ffffff";
    }
}
```

Rys. 4.15. Przykładowy motyw dziedziczący po klasie bazowej

```
18 references
public static class CurrentTheme
{
    7 references
    public static string BackgroundColor1 { get; set; }
    2 references
    public static string BackgroundColor2 { get; set; }
    1 reference
    public static string BackgroundColor3 { get; set; }
    2 references
    public static string ButtonColor { get; set; }
    8 references
    public static string TextColor { get; set; }
    3 references
    public static void SetTheme(ThemeModel theme)
    {
        BackgroundColor1 = theme.BackgroundColor1;
        BackgroundColor2 = theme.BackgroundColor2;
        BackgroundColor3 = theme.BackgroundColor3;
        ButtonColor = theme.ButtonColor;
        TextColor = theme.TextColor;
    }
}
```

Rys. 4.16. Klasa przechowująca aktualny motyw ustawiana na podstawie modelu abstrakcyjnego

Motyw to klasa dziedzicząca po klasie bazowej z odpowiednio skonfigurowanym konstruktorem, dzięki takiej konstrukcji w razie potrzeby dodania kolejnych motywów nie ma potrzeby modyfikacji klasy głównej. Wystarczy dodać dowolną liczbę nowych motywów w postaci klas oraz przypisać obiekt nowo powstałej klasy w klasie inicjalizującej.

Kolejnym krokiem było stworzenie klasy przechowującej aktywny w danym momencie motyw (rysunek 4.17).

```
18 references
public static class CurrentTheme
{
    7 references
    public static string BackgroundColor1 { get; set; }
    2 references
    public static string BackgroundColor2 { get; set; }
    1 reference
    public static string BackgroundColor3 { get; set; }
    2 references
    public static string ButtonColor { get; set; }
    8 references
    public static string TextColor { get; set; }
    3 references
    public static void SetTheme(ThemeModel theme)
    {
        BackgroundColor1 = theme.BackgroundColor1;
        BackgroundColor2 = theme.BackgroundColor2;
        BackgroundColor3 = theme.BackgroundColor3;
        ButtonColor = theme.ButtonColor;
        TextColor = theme.TextColor;
    }
}
```

Rys. 4.17. Klasa statyczna przechowująca motyw

Następnie należało stworzyć metodę inicjalizującą ustawiającą motyw (rysunek 4.18) oraz wywołać ją w momencie ładowania aplikacji (rysunek 4.19).

```
1 reference
public class Initializer
{
    1 reference
    public static void LoadTheme()
    {
        //There will be a loading from deserialized object(setup file) here in the future
        CurrentTheme.SetTheme(new DarkTheme());
    }
}
```

Rys. 4.18. Klasa inicjalizująca

```
2 references
public App()
{
    Initializer.LoadTheme();
```

Rys. 4.19. Klasa inicjalizująca - wywołanie

Następnie został stworzony mechanizm aktualizujący widok po każdym załadowaniu, przedstawiony na rysunku 4.20.

```
1 reference
private void SetLayout()
{
    ((StackLayout)FindByName("Logo")).BackgroundColor = Color.FromHex(CurrentTheme.BackgroundColor1);
    ((StackLayout)FindByName("Content")).BackgroundColor = Color.FromHex(CurrentTheme.BackgroundColor2);

    for (int i = 1; i <= 12; i++)
        ((Label)FindByName($"Label{i}")).TextColor = Color.FromHex(CurrentTheme.TextColor);
```

Rys. 4.20. Aktualizacja widoku

Na koniec do widoku dodano przełącznik sterujący zmianami motywu wraz z mechanizmem działania (rysunki 4.21 oraz 4.22).

```
<Grid
    VerticalOptions="Start"
    x:Name="switch"
    Padding="20, 20, 20, 20">

    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
    </Grid.RowDefinitions>
    <Label
        x:Name="Label2"
        VerticalTextAlignment="Center"
        Grid.Row="0"
        Grid.Column="0"
        Text="Dark mode"
        FontSize="18">
    </Label>
    <Switch
        IsToggled="True"
        Toggled="Switch_Toggled"
        OnColor="#878385"
        ThumbColor="AliceBlue"
        Grid.Row="0"
        Grid.Column="1"
        VerticalOptions="Center"/>
</Grid>
```

Rys. 4.21. Kod XAML przełącznika

```
0 references
private void Switch_Toggled(object sender, ToggledEventArgs isToggled)
{
    if (isToggled.Value)
        CurrentTheme.SetTheme(new DarkTheme());
    else
        CurrentTheme.SetTheme(new LightTheme());

    UpdateLayout();
}
```

Rys. 4.22. Mechanizm działania przełącznika

Rysunki 4.23 oraz 4.24 przedstawiają wygląd przełącznika oraz zmiany wyglądu całego widoku po jego użyciu.

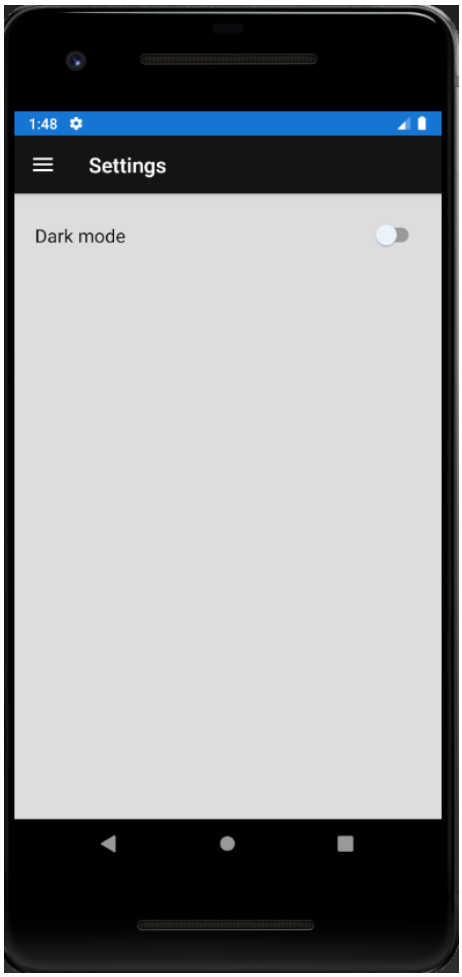
Rys. 4.23. Włączenie trybu ciemnego

5. Testowanie

5.1. Menu

Aby testowanie było bezproblemowe i kompleksowe należy przetestować każdą część aplikacji stopniowo zwiększając stopień skomplikowania. Pierwszym testem będzie działanie menu, składa się on z następującego scenariusza:

1)Otworzenie aplikacji
2)Sprawdzenie czy strona główna to widok zaimplementowany w klasie MainPage
3)Sprawdzenie czy każda opcja w menu pokrywa się z zaimplementowanym widokiem
4)Ustawienie aplikacji na działanie w tle, ponowne włączenie, i sprawdzenie czy otwiera się na ostatniej przeglądanej stronie
5)Przełączanie zakładek używając multitoucha



Rys. 4.24. Wyłączenie trybu ciemnego

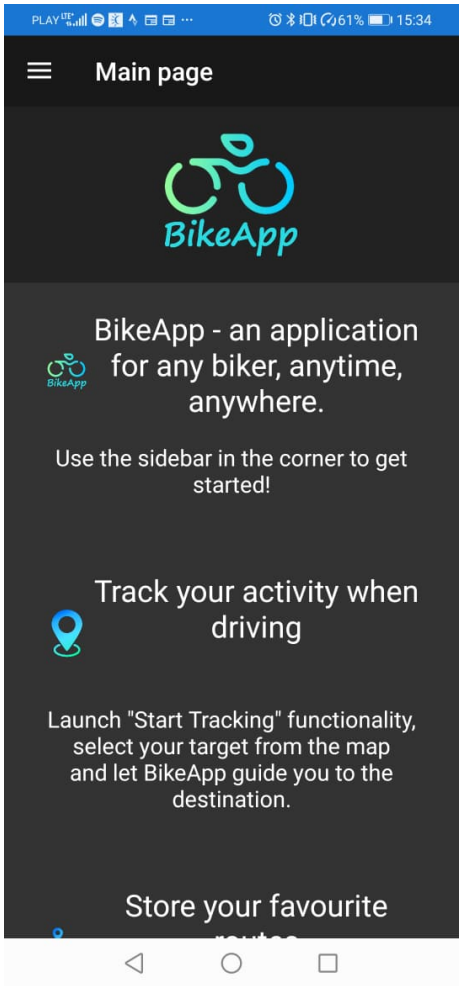
Wyniki: Aplikacja działa poprawnie, nie pojawia się żaden bug

6. Podręcznik użytkownika

6.1. Czym jest BikeApp?

BikeApp jest aplikacją stworzoną dla rowerzystów wszelkiego rodzaju, od osób jeżdżących rekreacyjnie, przez turystów aż po zawodowych kolarzy (rysunek 6.1). Pozwala na zapisanie w pamięci telefonu różnych danych, które mogą być mu potrzebne, takich jak często odbywane trasy oraz dane pomiarowe. Oferuje przy tym prostą obsługę tych udogodnień oraz możliwość spersonalizowania ustawień.

Niniejsza instrukcja zawiera informacje odnośnie korzystania z aplikacji, a w szczególności - komponentów, na które się ona składa. Jedna z podsekcji poświęcona została także znanym problemom, na które użytkownik może natrafić, korzystając z BikeApp.



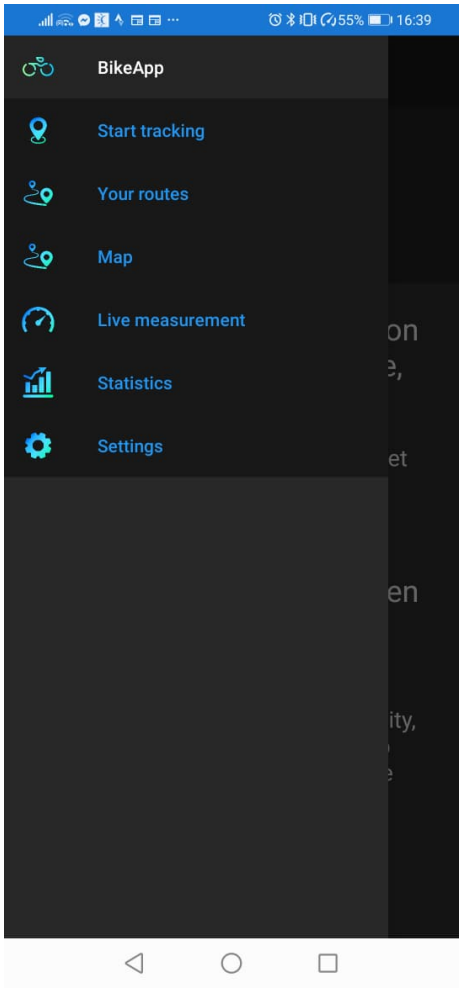
Rys. 6.1. BikeApp - strona główna

6.2. Dostęp do menu głównego oraz spis podstron

Aplikacja BikeApp podzielona jest na podstrony, zaś dostęp do nich możliwy jest za pośrednictwem menu głównego.

Menu główne jest otwierane przy użyciu znajdującej się w lewym górnym rogu ekranu ikony trzech poziomych linii. Jżeli zamiast niej widoczna jest strzałka, wówczas oznacza ona cofnięcie się do poprzedniej podstrony i należy ją wybierać i cofać się aż do pojawienia się ikony otwarcia menu głównego.

Wybranie opcji na menu głównym (rysunek 6.2) spowoduje otwarcie właściwej podstrony poświęconej danej funkcji. Tematy obsługi poszczególnych podstron oraz opisu funkcji rozwinięto w dalszych podrozdziałach.



Rys. 6.2. BikeApp - menu główne

6.3. Mechanizm śledzenia trasy

Tryb śledzenia trasy pozwala na zapisanie ścieżki, którą użytkownik przebędzie w trakcie jego działania. Powstała w ten sposób trasa może następnie zostać wyświetlona na mapie bądź zapisana w pamięci telefonu w celu późniejszego przeglądania.

Aby uruchomić podstronę śledzenia trasy, należy rozwinąć menu główne i wybrać opcję “Start Tracking”.

Tryb śledzenia trasy jest uruchamiany poprzez wybranie przycisku “Start tracking” na podstronie (rysunek 6.3). Jego włączenie potwierdzi stosowny wyskakujący komunikat. Ponowne użycie tego samego przycisku zatrzymuje działanie trybu śledzenia trasy, a następnie wyświetla monit o nadanie trasie nazwy, po czym zostaje ona zapisana przez aplikację.

W trakcie śledzenia trasy możliwe jest wykonywanie innych czynności na telefonie, w tym zamknięcie okna aplikacji i wyjście do ekranu głównego systemu telefonu.

6.4. Zarządzanie zapisanymi trasami

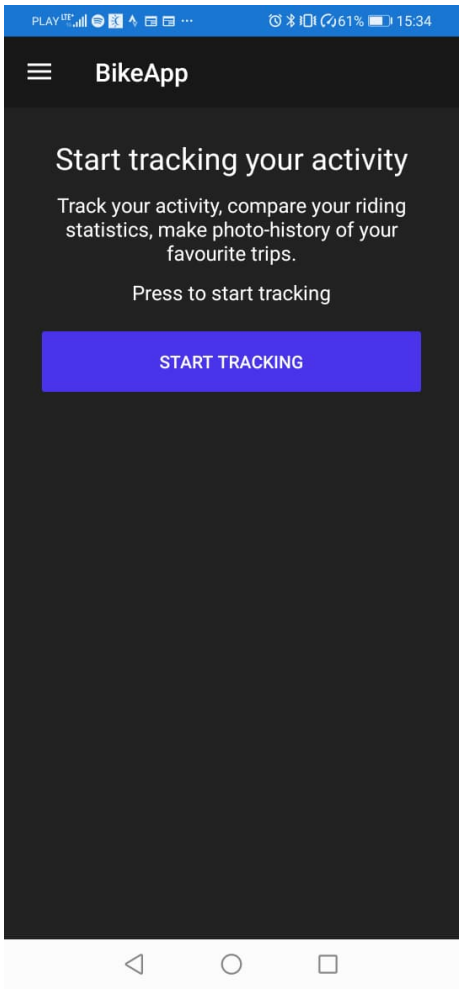
Po zakończeniu śledzenia danej trasy, jest ona zapisywana pod nazwą wskazaną przez uzytkownika. Lista tras oraz parametry każdej z nich dostępne są z poziomu specjalnej podstrony.

Aby obejrzeć zapisane trasy, należy rozwinąć menu główne i wybrać opcję “Your Routes”.

Każda pozycja na liście może zostać zaznaczona, co przeniesie użytkownika do ekranu z informacjami na temat wybranej trasy (rysunek 6.4). Aby powrócić do ekranu ze spisem zapisanych tras, należy użyć strzałki znajdującej się w lewym górnym rogu. Działanie przycisku “Display on Map” opisano w podrozdziale “Wyświetlanie trasy na mapie” niniejszej instrukcji.

6.5. Wyświetlanie trasy na mapie

BikeApp zawiera podstronę z Mapą Google, dzięki której zapisane w pamięci telefonu trasy można obejrzeć w trybie graficznym. Będzie ona widoczna jako seria punktów połączonych prostymi liniami.



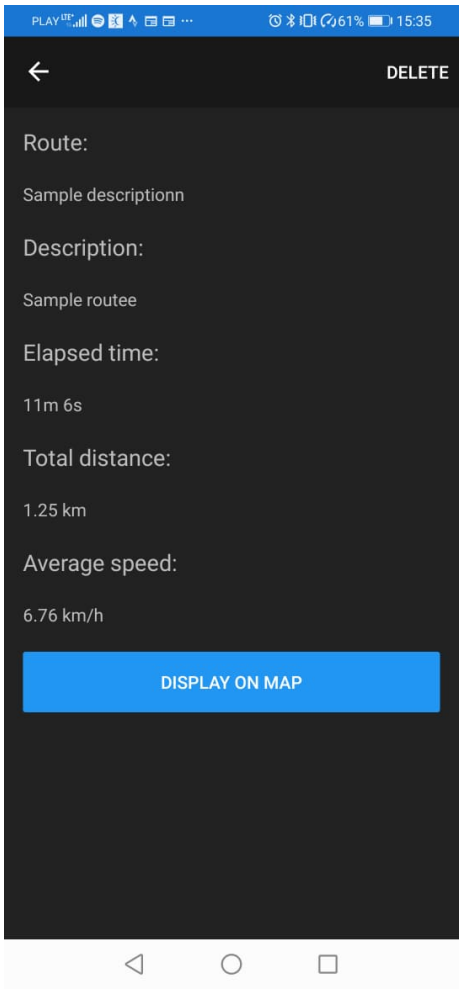
Rys. 6.3. BikeApp - podstrona śledzenia trasy

Istnieją dwa sposoby na otwarcie Mapy Google wewnątrz aplikacji. Pierwszym jest wybranie opcji "Map" z menu głównego, drugim zaś - uruchomienie podstrony z listą tras, wybranie trasy oraz użycie przycisku "Display on Map". W przypadku sukcesu, widok będzie podobny, jak na rysunku 6.5.

Przesuwanie oraz regulacja przybliżenia widoku na mapę odbywają się w sposób standardowy dla Map Google uruchamianych na urządzeniach mobilnych. Aplikacja nakłada również na mapę linie reprezentujące wybraną trasę.

6.6. Przeglądanie danych pomiarowych

Aplikacja BikeApp może dokonywać pomiarów w trakcie jazdy. Mierzone są takie parametry jak kąty nachylenia urządzenia, przeciążenia oraz wystąpienia skoków



Rys. 6.4. BikeApp - szczegóły trasy

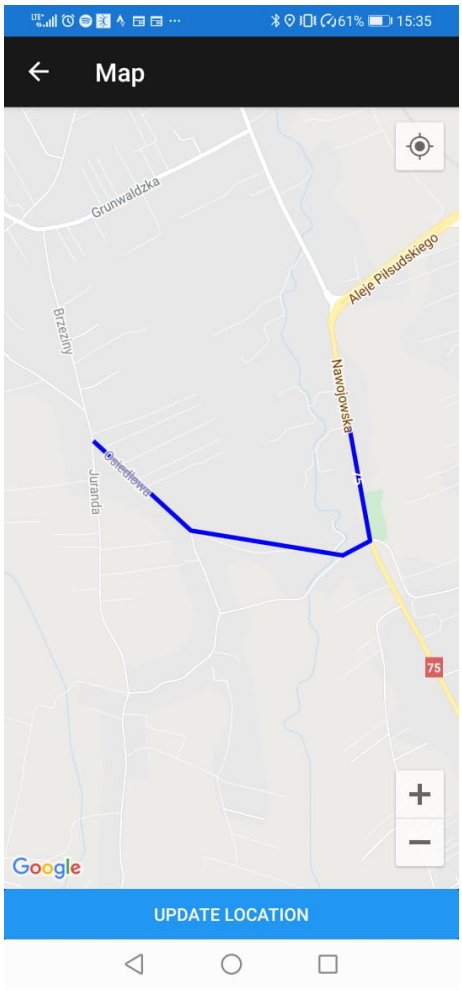
przeciążeń, wynikających przykładowo z natrafienia na przeszkodę. Dane te mogą być wyświetlane na bieżąco bądź zbierane i przechowywane jako statystyki użytkownika.

6.6.1. Dane chwilowe

Aby uzyskać dostęp do danych wyświetlanych w momencie pobrania, należy z poziomu menu głównego wybrać pozycję "Live measurement". Użytkownikowi ukaże się podstrona wyglądająca tak, jak na rysunku 6.6.

Pomiary danych są domyślnie zatrzymane po uruchomieniu aplikacji. Można je rozpocząć, korzystając z przełącznika odpowiadającego pozycji "Accelerometer". Poniżej przełączników będą widoczne zmierzone przez aplikację wartości.

Drugi przełącznik, opisany nazwą "Shake detection", obsługuje wykrywanie skoków



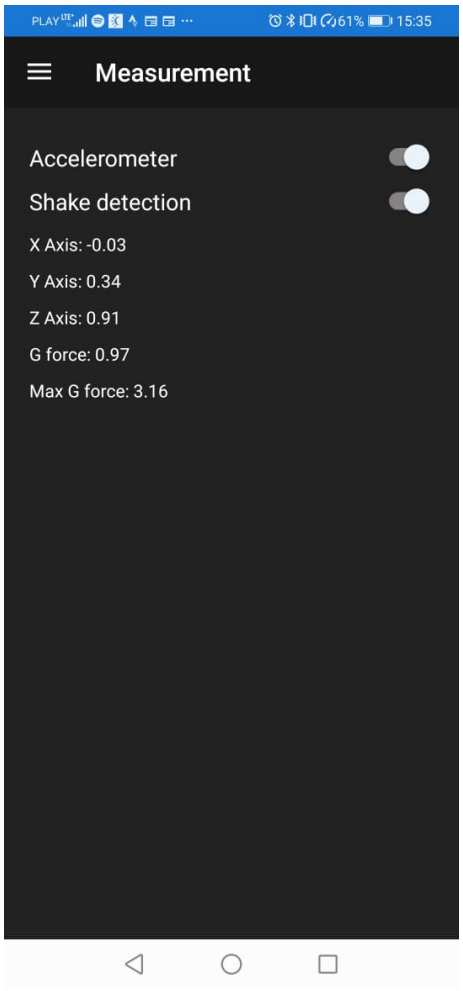
Rys. 6.5. BikeApp - Mapa Google

przeciążenia. Gdy jest ono aktywne, każdy moment, w którym pomiar przeciążenia wykaże odpowiednio wysoką wartość, będzie odnotowywany poprzez wyskakujące okienko. Wykrywania skoków przeciążenia nie można uruchomić, jeśli wyłączone jest także pobieranie danych chwilowych.

6.6.2. Dane gromadzone w czasie (statystyki)

Aby zobaczyć dane zbierane w dłuższym przedziale czasu, należy otworzyć menu główne i wybrać opcję “Statistics”.

Podstrona statystyk (rysunek 6.7) przedstawia różne rodzaje danych zgromadzonych przez aplikację w czasie, gdy tryb śledzenia trasy był aktywny. Obejmuje zarówno informacje o wartościach minimalnych i maksymalnych, jak również średnich.



Rys. 6.6. BikeApp - Pomiary chwilowe

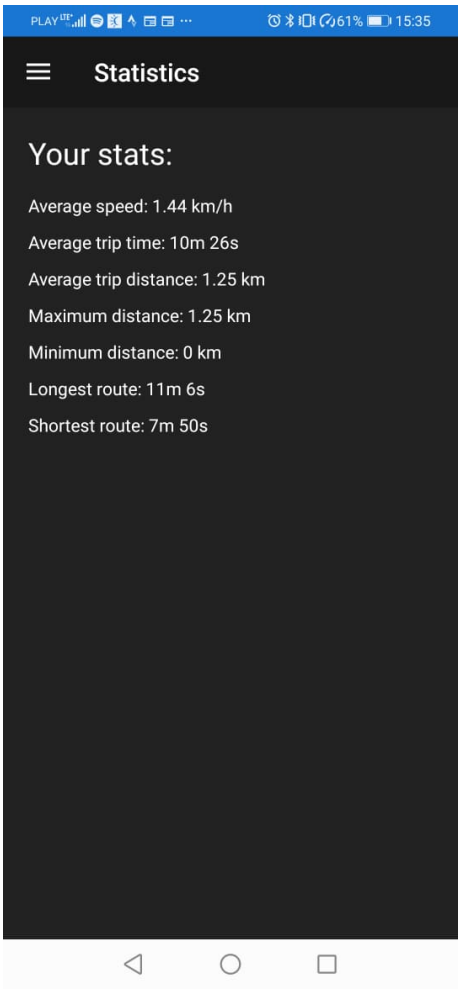
6.7. Dostosowanie ustawień

Ustawienia aplikacji dostępne są za pośrednictwem menu głównego. Po jego otwarciu należy wybrać pozycję “Settings”.

Poszczególne opcje dostępne są w postaci przełączników, które można aktywować lub deaktywować poprzez ich użycie. Każda zmiana ustawień zatwierdzana jest natychmiast po jej wystąpieniu.

6.8. Częste problemy i błędy

Lorem ipsum...



Rys. 6.7. BikeApp - Podstrona statystyk

Spis rysunków

Spis tabel