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. Zamówienie aplikacji przez 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.

Niżej przedstawiam proponowany ekran główny aplikacji:

Rys. 2.1. Propozycja ekranu głównego

3. Projektowanie

3.1. Narzędzia

Do wykonania aplikacji użyte zostaną następujące narzędzia:

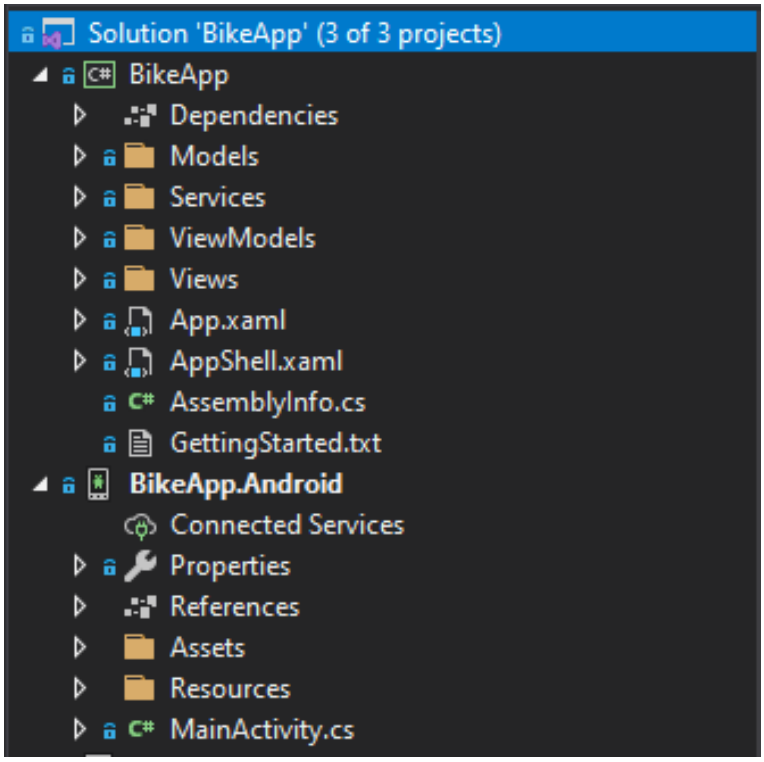
- *Visual Studio 2019*
- *Xamarin Forms v5.0.0.2125*
- *GitHub*
- *Nuget packages*
- *Android Emulator(Pixel 2 Pie 9.0)*

Szkic strony głównej:

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 prezentuje się następująco:



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 typu background color, font-size, padding, textcolor itp. aby wystylizować układ widoku.

Fragment 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. Zmiany styli

-Stworzyć widok główny na podstawie języka XML oraz metody w języku C# informującej o kliknięciu w przycisk.

Fragment modyfikacji:

```
<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 android w języku java

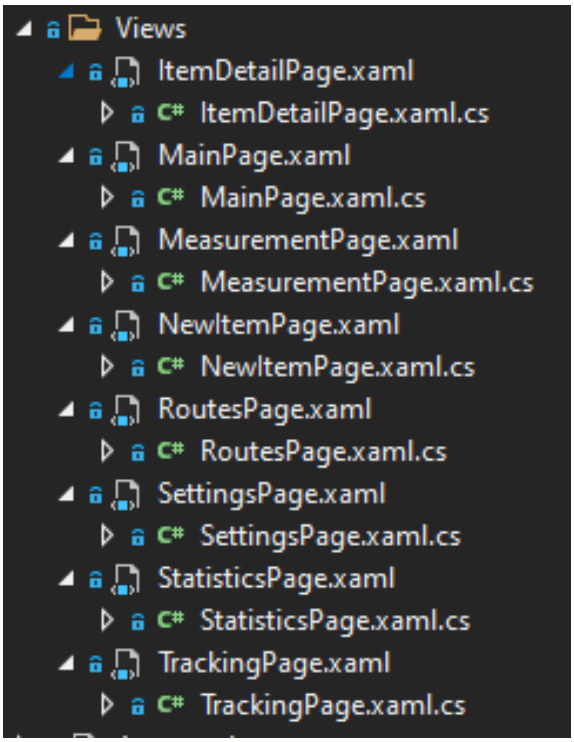
Zmiany widoczne w kodzie:

```
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

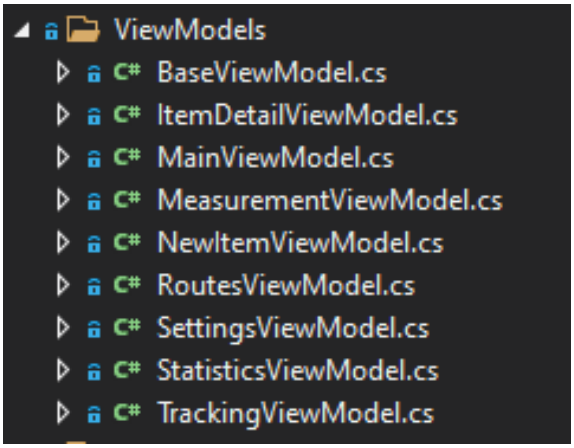
4.3. Dodanie oraz wystylizowanie elementów menu

Na początku należało utworzyć odpowiednie widoki aby stworzyć spójny szkielet aplikacji



Rys. 4.4. Szkielet widoków

Następnie należało stworzyć odpowiadające im modele widoków



Rys. 4.5. Szkielet modeli widoków

Potem należało ustawić odpowiednie parametry tak aby połączyć poszczególny widok z jego modelem

```
<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. Binding ModelView z View

Kolejnym krokiem było ustawienie odpowiednich opcji flyoutmenu:

```
<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 należało stworzyć oraz dodać odpowiednie ikony wraz z tytułami do każdego widoku

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

Rys. 4.8. Dodanie styli

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

Rys. 4.9. Dodanie styli

```
<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

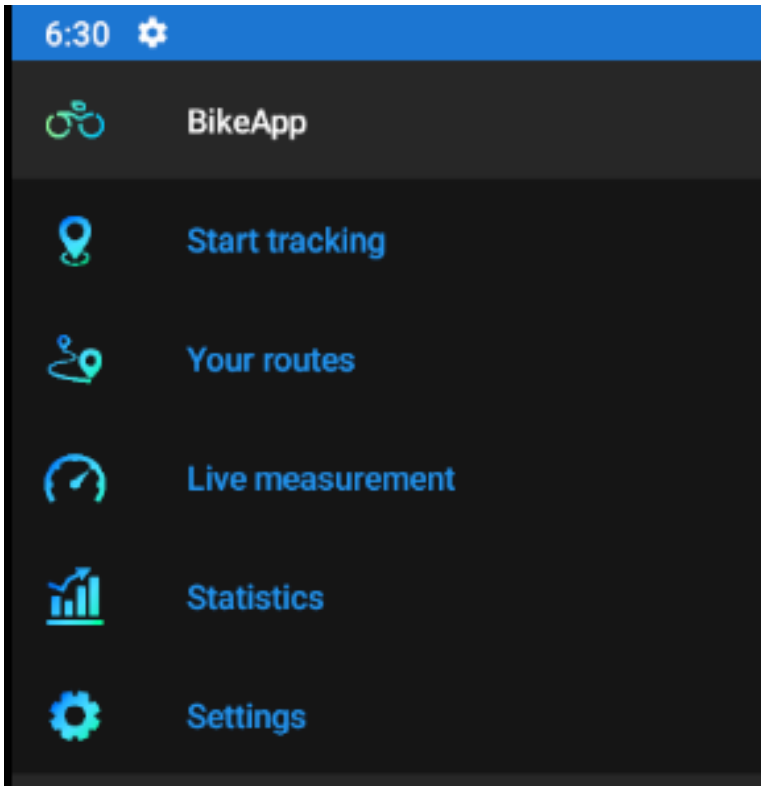
Na koniec należało wszystko wystylizować oraz poprawić ewentualne błędy w kodzie

```
<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

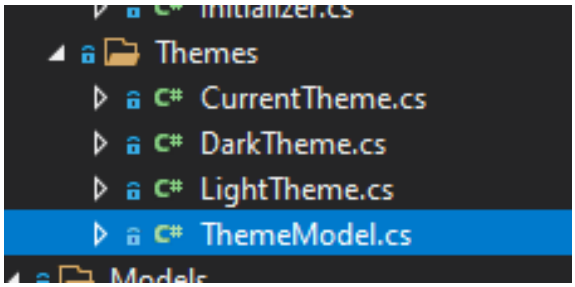
Efekt finalny:



Rys. 4.12. Finalizacja

4.4. Dodanie motywu jasnego i ciemnego

Na początku należało utworzyć odpowiednie klasy przechowujące kolory



Rys. 4.13. Układ klas

```
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. Abstrakcyjna klasa bazowa

```
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 ustawiony motyw

```
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, która ustawia motyw oraz wywołać ją w momencie ładowania aplikacji.

```
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

Potem należało napisać mechanizm aktualizujący widok po każdym załadowaniu

```
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

Ostatnim krokiem było utworzenie switcha oraz jego mechanizmu działania

```
<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. Widok XAML dla switcha

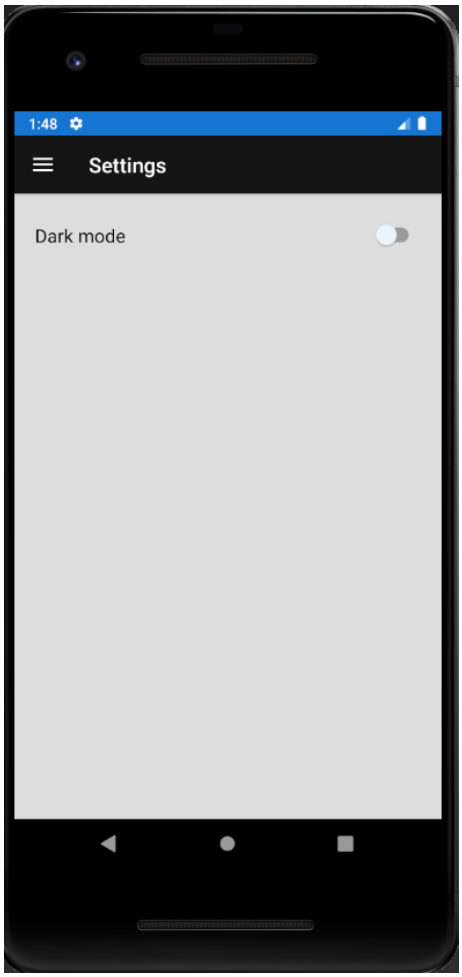
```
0 references
private void Switch_Toggled(object sender, ToggledEventArgs isToggled)
{
    if (isToggled.Value)
        CurrentTheme.SetTheme(new DarkTheme());
    else
        CurrentTheme.SetTheme(new LightTheme());

    UpdateLayout();
}
```

Rys. 4.22. Mechanizm działania switcha

Efekt finalny:

Rys. 4.23. Dark mode włączony



Rys. 4.24. Dark mode wyłączony

5. Testowanie

6. Podręcznik użytkownika

Spis rysunków

Spis tabel