WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: **INFORMATYKA**
Specjalność: **Technologie internetowe i mobilne**

Izabella Nosek
Nr albumu studenta: w71345

Mobilna aplikacja pogodowa „Pogodynka”

Prowadzący: dr Marek Jaszuk

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Rzeszów 2025

Spis treści

Wstęp

Platforma .NET MAUI oferuje wiele możliwości w projektowaniu wieloplatformowych aplikacji. Aplikacja *Pogodynka* stara się je efektywnie wykorzystać. Łatwy i przejrzysty interfejs pozwala na wygodne użytkowanie i zapoznanie się z bieżącą pogodą oraz prognozą na najbliższe dni w wybranej przez użytkownika miejscowości.

W *Pogodynce* można znaleźć niezbędne informacje o pogodzie w dwóch językach – polskim i angielskim, ponieważ dostosowuje się ona do języka systemowego w telefonie. Dodatkowym udogodnieniem jest dopasowanie motywu aplikacji do bieżącego motywu telefonu (tryb jasny, tryb ciemny) oraz opcja automatycznego pobrania bieżącej lokalizacji.

1. Opis projektu

1.1. Założenia i cele projektu, wykorzystane technologie

Założenia i cele projektu

Projekt ma na celu stworzenie przystępnej aplikacji pogodowej, dzięki której użytkownik może sprawdzić prognozę pogody w każdym momencie. Aplikacja ma umożliwiać sprawdzenie bieżącej pogody w wybranej przez użytkownika lokalizacji oraz prognozę na kolejnych 5 dni. Dane pogodowe pobierane są z zewnętrznego API. Wymagane jest połączenie z Internetem. Aplikacja oferuje także automatyczne pobranie bieżącej lokacji.

Celem projektu jest:

- Stworzenie intuicyjnej aplikacji mobilnej do sprawdzania pogody
- Zapewnienie możliwości sprawdzenia bieżącej pogody, jak i prognozy
- Obsługa modułu GPS do pobrania lokalizacji
- Zaprojektowanie przyjaznego i czytelnego interfejsu użytkownika
- Dostosowanie do trybu jasnego/ciemnego urządzenia
- Dostosowanie języka aplikacji do języka systemowego (EN i PL)
- Dostosowany ekran ładowania i ikona aplikacji

Wykorzystane technologie

Projekt został utworzony w .NET MAUI 9, korzystając z *Visual Studio 2022* i *Android Studio* (Pixel 7 Android 15). Dane pogodowe dostarcza open-source API *Open-Meteo*.

Animacje zastosowane w aplikacji pochodzą z darmowej strony *LottieFiles* https://lottiefiles.com/. Grafika ikony i ekranu ładowania została utworzona w Adobe Illustrator, bazując na użytych grafikach animacji pogody.

Dodatkowo, do obsługi interfejsu *AddINotifyPropertyChangedInterface* zainstalowano package *Fody*. *Fody* to narzędzie do automatycznej modyfikacji skompilowanego kodu .NET. Umożliwia eliminację powtarzalnego kodu, np. implementacji *INotifyPropertyChanged*.

Do obsługi animacji .json zainstalowano package *SkiaSharp.Extended.UI.Maui*, zawierający kontrolery *SkiaSharp* dla MAUI. *SkiaSharp* to system grafiki 2D dla platformy .NET i języka C#, który rysuje grafikę wektorową 2D, mapy bitowe i tekst.

1.2. Wymagania funkcjonalne i niefunkcjonalne

Wymagania funkcjonalne:

1. *Wyświetlanie aktualnej pogody* - aplikacja pokazuje aktualną temperaturę, wilgotność, ciśnienie, prędkość i kierunek wiatru oraz warunki pogodowe (np. bezchmurnie)
2. *Prognoza pogody* - użytkownik może sprawdzić prognozę pogody na kolejnych 5 dni
3. *Wyszukiwanie lokalizacji* - aplikacja umożliwia wyszukiwanie pogody dla danego miasta oraz pobranie bieżącej lokalizacji wykorzystując moduł GPS urządzenia.

4. *Dopasowanie do motywu telefonu* - aplikacja dostosowuje się do trybu jasnego/ciemnego
5. *Dostosowany język aplikacji* – aplikacja obsługuje język angielski i polski, zależnie od ustawień systemu. W przypadku innego języka, stosuje język uniwersalny (angielski)
6. *Identyfikacja wizualna* - aplikacja posiada własną ikonę i grafikę ładowania.

Wymagania niefunkcjonalne:

1. *Wydajność* - aplikacja powinna ładować dane pogodowe w mniej niż 3 sekundy przy normalnym połączeniu internetowym
2. *Responsywność* - interfejs użytkownika powinien być płynny i reagować bez opóźnień
3. *Bezpieczeństwo* - aplikacja nie powinna gromadzić więcej danych niż potrzebne, lokalizacja musi być wykorzystywana zgodnie z polityką prywatności.
4. *Skalowalność* - aplikacja powinna być przygotowana na przyszłe rozbudowy
5. *Zgodność* - aplikacja musi działać poprawnie na urządzeniach z Androidem w wersji min. 11.0 i wyżej.
6. *Łatwość obsługi* - intuicyjny interfejs, który nie wymaga szkolenia ani instrukcji obsługi.
7. *Stabilność* - aplikacja nie powinna się zawieszać ani nagle kończyć działania, powinna posiadać optymalizowane zużycie baterii i zasobów

2. Struktura aplikacji

W projekcie zastosowano podział MVVM (Model-View-ViewModel) dla lepszej czytelności kodu – widoki stron i odpowiednie im klasy zostały umieszczone w dedykowanych folderach. Dodatkowo utworzono osobny folder dla konwerterów.

Pliki utworzone w projekcie:

- *PogodaView* – pliki określające widok strony głównej (frontend)
- *PogodaViewModel* – główny plik do obsługi wyszukania lokalizacji i pobrania pogody oraz połączenia danych z widokiem
- *PogodaModel* – plik zwierający strukturę obiektów API
- *KodPogodaConverter* – konwerter przypisujący odpowiedni opis pogody pod kod pogody otrzymany z API
- *KodGrafikaConverter* - konwerter przypisujący odpowiednią grafikę pogody pod kod pogody otrzymany z API
- *AppResources* – pliki przechowujące tłumaczenie tekstów aplikacji
- *Colors* – plik przechowujący kolorystykę aplikacji
- *Styles* – plik przechowujący style aplikacji

2.1. Układ strony głównej

Ekran aplikacji został utworzony w siatce (grid) i podzielony na trzy regiony: wyszukiwarka lokalizacji, bieżące informacje pogodowe wyszukanej lokalizacji i prognoza pogody. Dodatkowo zaimplementowano *ActivityIndicator*, obrazujący ekran ładowania podczas pobierania danych pogodowych.

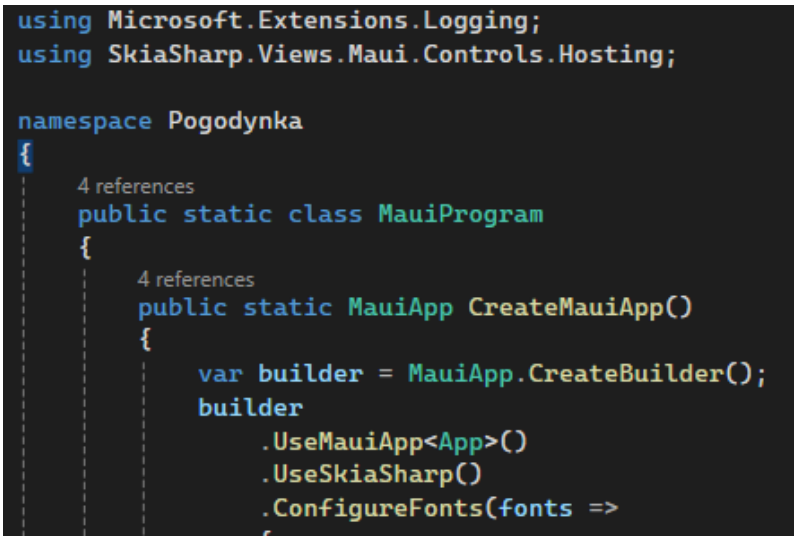
Zdj. 1 Schemat strony głównej - podział na regiony

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Pogodynka.MVVM.Views.PogodaView"
             xmlns:SkiaSharp="clr-namespace:SkiaSharp.Extended.UI.Controls;assembly=SkiaSharp.Extended.UI"
             xmlns:strings="clr-namespace:Pogodynka.Resources.Strings"
             xmlns:converter="clr-namespace:Pogodynka.Converters"
             Title="Pogodynka">

    <ContentPage.Resources>
        <converter:KodPogodaConverter x:Key="KodPogodaConverter"/>
        <converter:KodGrafikaConverter x:Key="KodGrafikaConverter"/>
    </ContentPage.Resources>

    <Grid RowDefinitions=".06*, .5*, .33*" Style="{StaticResource MainGrid}">
        <!--#region Searchbar -->
        <Border...>
        <!--#endregion-->

        <!--#region Aktualna pogoda -->
        <Grid Grid.Row="1" RowDefinitions=".2*,.6*, .15*" ColumnDefinitions=".6*, .4*" IsVisible="{Binding IsVisible}"...
        <!--#endregion-->

        <!--#region Prognoza -->
        <CollectionView Grid.Row="2" ItemsSource="{Binding PogodaData.daily_single}"...>
        <!--#endregion-->

        <!--ActivityIndicator-->
        <Grid Margin="-8" Grid.RowSpan="3" RowDefinitions="*, *, *, *, *" IsVisible="{Binding IsLoading}" Style="{Static
            <ActivityIndicator Grid.Row="1" IsRunning="True"/>
        </Grid>
    </Grid>
</ContentPage>
```

Źródło: Praca własna w Visual Studio 2022

Do obsługi animacji w formacie *json* niezbędne było zainstalowanie rozszerzenia *SkiaSharp.Extended.UI.*Maui, które wprowadza nowy widok *SKLottieView*. Aby aktywować SkiaSharp, należy dodać namespace oraz metodę *UseSkiaSharp()* w builderze aplikacji (klasa *MauiProgram*).

Zdj. 2 Implementacja SkiaSharp w projekcie



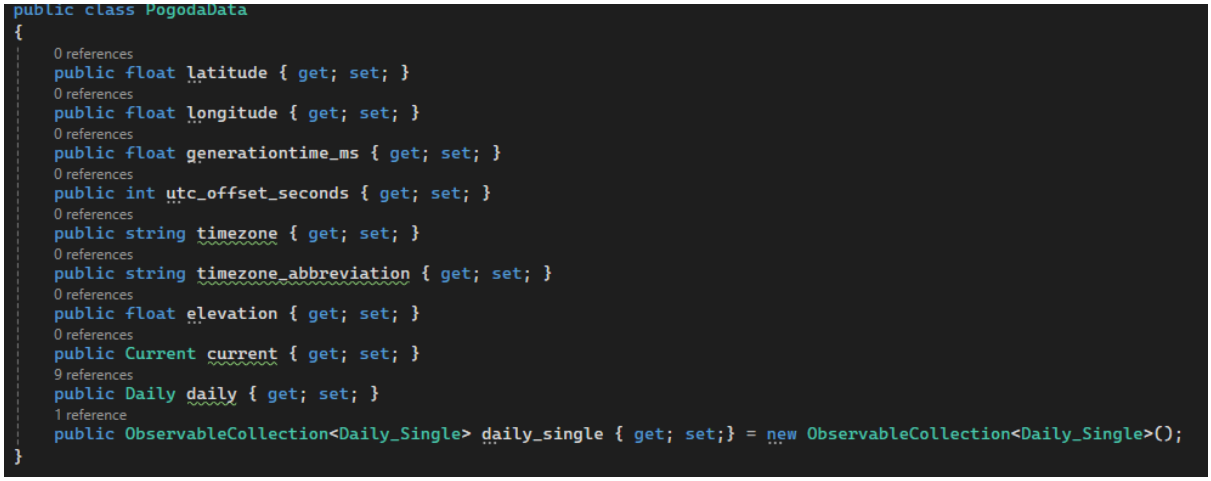
Źródło: Praca własna w Visual Studio 2022

Prognoza pogody została zobrazowana za pomocą *CollectionView* w widoku poziomym – użytkownik może przewijać kolejne widoki prognozy. Każdy widok zawiera wszystkie informacje pogodowe (więcej w rozdziale *Prezentacja Aplikacji*).

2.2. Klasa *PogodaData* – obiekty API

Plik *PogodaData.cs* zawiera klasy obiektów API (główny obiekt *PogodaData*, klasę *Current* z danymi bieżącej pogody i *Daily* z prognozami) oraz dodatkową klasę *DailySingle* (utworzoną na wzór klasy *Daily*).

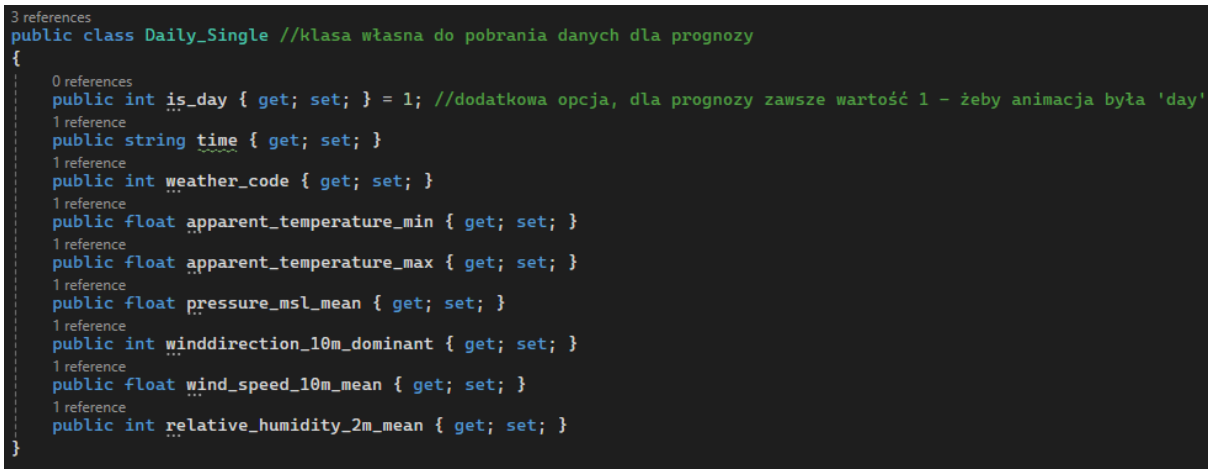
Zdj. 3 Struktura klasy *PogodaData*



Źródło: Praca własna w Visual Studio 2022

Klasa *DailySingle* służy do pobierania danych konkretnego dnia z prognozy, ponieważ klasa *Daily* zawiera dane tylko w postaci tablicy. Przypisano także wartość *isDay* jako 1, ponieważ API samo w sobie nie zawiera wartości *isDay* dla prognozy pogody kolejnych dni, a wartość 0 lub 1 jest wymagana, by zastosować konwerter graficzny. Klasę *DailySingle* przypisano jako dynamiczną kolekcję danych typu *ObservableCollection* do głównej klasy *PogodaData*.

Zdj. 4 Struktura klasy Daily_Single



```
3 references
public class Daily_Single //klasa własna do pobrania danych dla prognozy
{
    0 references
    public int is_day { get; set; } = 1; //dodatkowa opcja, dla prognozy zawsze wartość 1 - żeby animacja była 'day'
    1 reference
    public string time { get; set; }
    1 reference
    public int weather_code { get; set; }
    1 reference
    public float apparent_temperature_min { get; set; }
    1 reference
    public float apparent_temperature_max { get; set; }
    1 reference
    public float pressure_msl_mean { get; set; }
    1 reference
    public int winddirection_10m_dominant { get; set; }
    1 reference
    public float wind_speed_10m_mean { get; set; }
    1 reference
    public int relative_humidity_2m_mean { get; set; }
}
```

Źródło: Praca własna w Visual Studio 2022

2.3. Klasa *PogodaViewModel* – obsługa zapytań API i widoczności danych

Klasa *PogodaViewModel* zawiera w sobie obsługę wyszukiwania lokalizacji, pobierania danych o pogodzie oraz ustawień widoczności pól startowych. Stosuje właściwość *AddINotifyPropertyChangedInterface* do pobierania informacji o zmianach po stronie użytkownika. Poniżej rozpisano elementy klasy i ich zastosowanie.

Zmienne początkowe:

- *PogodaData* – obiekt klasy PogodaData, przechowuje dane pogodowe pobrane z API
- *PlaceName* – wartość typu string, do której przypisywana jest nazwa miejscowości wpisana przez użytkownika w wyszukiwarce
- *PlaceNameFound* – wartość typu string służąca do przypisana znalezionej lokacji i zwrócenia jej do widoku użytkownika
- *DateNow* – wartość typu DateTime, zawierająca bieżącą datę i godzinę w momencie utworzenia ViewModelu
- *IsVisible* – wartość typu bool, określająca widoczność szczegółów pogody w interfejsie użytkownika (ukryta na początku).
- *IsLoading* – wartość typu bool, kontrolująca widoczność wskaźnika ładowania (ActivityIndicator) podczas pobierania danych
- *isBusy* – wartość pomocnicza typu bool, chroniąca przed wielokrotnym uruchomieniem wyszukiwania (np. przez szybkie kliknięcie)
- *HttpClient* – instancja klasy HttpClient, służąca do komunikacji z zewnętrznym API (pobierania danych pogodowych). Jest inicjalizowana w konstruktorze klasy *PogodaViewModel*, aby umożliwić dostęp do danych pogodowych zaraz po utworzeniu obiektu tego ViewModelu.

Zdj. 5 Zarys klasy *PogodaViewModel* – podział na regiony

```
namespace Pogodynka.MVVM.ViewModels
{
    [AddINotifyPropertyChangedInterface]
    2 references
    public class PogodaViewModel
    {
        Initial variables

        1 reference
        public PogodaViewModel()
        {
            //inicjacja klienta do api
            HttpClient = new HttpClient();
        }

        Get current location

        Search location

        Get weather info
    }
}
```

Źródło: Praca własna w Visual Studio 2022

2.3.1. Pobieranie współrzędnych geograficznych

Do pobrania współrzędnych geograficznych danej lokalizacji utworzono funkcję *GetCoordinatesAsync()* przyjmującą nazwę lokalizacji jako argument. Funkcja ta wywołuje asynchroniczną metodę geokodowania *Geocoding.Default.GetLocationsAsync()*, która przekłada podany adres na współrzędne geograficzne. Następnie zwraca pierwszą lokalizację ze zwracanej przez metodę geokodowania listy lokalizacji.

Zdj. 6 Pobieranie współrzędnych geograficznych

```
//Pobranie współrzędnych znalezionej lokalizacji
1 reference
private static async Task<Location> GetCoordinatesAsync(string address)
{
    IEnumerable<Location> locations = await Geocoding.Default.GetLocationsAsync(address);
    Location? location = locations?.FirstOrDefault();

    return location;
}
```

Źródło: Praca własna w Visual Studio 2022

2.3.2. Automatyczne pobieranie lokalizacji urządzenia

Aby skorzystać z modułu GPS zmodyfikowano plik *AndroidManifest.xml* aby włączyć uprawnienia ACCESS_FINE_LOCATION oraz ACCESS_COARSE_LOCATION. Do pobrania danych o lokalizacji wykorzystano wbudowany w .NET MAUI interfejs *IGeolocation*, wprowadzający API pobierające dane lokalizacji. Dane połączono z wyszukiwarką na stronie głównej aplikacji stosując *Binding* oraz interfejs *ICommand*.

Do pobrania lokalizacji użytkownik musi wybrać ikonę lokalizacji w pasku wyszukiwania. Następnie wyzwalana jest komenda wywołująca task *GetWeatherFromDeviceLocationAsync()*, który sprawdza czy zgody na korzystanie z modułu GPS zostały udzielone. Jeśli zgody są poprawne, to pobiera dane lokalizacyjne i uruchamia akcję wyszukiwarki przypisując znalezioną miejscowość.

Zdj. 7 Pogląd na akcję pobrania bieżącej lokalizacji – przypisanie lokalizacji

```
if (status == PermissionStatus.Granted)
{
    var location = await Geolocation.GetLastKnownLocationAsync()
                   ?? await Geolocation.GetLocationAsync(new GeolocationRequest(GeolocationAccuracy.Medium));

    if (location != null)
    {
        var placemarks = await Geocoding.Default.GetPlacemarksAsync(location);
        var placemark = placemarks?.FirstOrDefault();

        var autoPlaceName = placemark?.Locality ?? placemark?.SubAdminArea ?? placemark?.AdminArea;

        if (!string.IsNullOrWhiteSpace(autoPlaceName))
        {
            PlaceNameFound = autoPlaceName;

            //Wrzucenie w wyszukiwarkę i uruchomienie komendy przypisanej do wyszukiwarki
            if (SearchCommand.CanExecute(autoPlaceName))
                SearchCommand.Execute(autoPlaceName);
```

Źródło: Praca własna w Visual Studio 2022

2.3.3. Pobieranie danych o wyszukiwanej lokalizacji

Komenda wyszukiwania pobiera nazwę z wyszukiwarki i sprawdza gotowość wyszukiwarki celem uniknięcia podwójnego wywołania. Następnie stosuje metodę *GetCoordinatesAsync()* i zwraca faktyczną nazwę szukanej miejscowości (np. „rzeszow” zmieni na „Rzeszów”).

Poprawna nazwa jest wówczas przekazywana do widoku użytkownika w miejscu nazwy miejscowości i wywołana zostaje metoda *GetWeather()*, która obsługuje zapytanie API i ewentualne błędy (stąd brak bloku *catch*).

Zdj. 8 Pogląd na metodę pobrania lokalizacji z wyszukiwarki

```
    if (isBusy)
        return;

    isBusy = true;

    try
    {
        PlaceName = searchText?.ToString();
        var location = await GetCoordinatesAsync(PlaceName);

        if (location == null)
        {
            await Shell.Current.DisplayAlert(AppResources.alertNotFoundTitle, AppResources.alertNotFoundMessage, "OK");
            return;
        }

        var placemarks = await Geocoding.Default.GetPlacemarksAsync(location); //pobranie prawdziwej nazwy lokacji
        var placemark = placemarks?.FirstOrDefault();
        PlaceNameFound = placemark.Locality ?? placemark.SubAdminArea ?? placemark.AdminArea ?? PlaceName;

        await GetWeather(location);
    }
    finally
    {
        isBusy = false;
```

Źródło: Praca własna w Visual Studio 2022

2.3.4. Obsługa zapytań API

Do obsługi zapytań API utworzono metodę *GetWeather()*, która przyjmuje lokalizację jako argument. Następnie konwertuje współrzędne (szerokość i długość geograficzną) na odpowiedni format i przekazuje je do zapytania API. Po otrzymaniu odpowiedzi z API, dane są deserializowane do obiektu *PogodaData*. Aby wyodrębnić dane dla poszczególnych dni

prognozy, wprowadzono dodatkową pętlę, która przypisuje otrzymane informacje do obiektów Daily_Single dla każdego dnia z osobna.

Konwersja współrzędnych na format, który jest odpowiedni dla zapytania API, została dodana w celu uniknięcia problemów wynikających z różnic w notacji liczbowej. W języku polskim stosuje się przecinek zamiast kropki jako separator dziesiętny, co powodowało błędy przy wysyłaniu zapytań API, gdy systemowy język był ustawiony na polski. W związku z tym zastosowano *InvariantCulture*, który zapewnia neutralny format liczbowy, niezależny od ustawień językowych systemu, używając standardowego formatu (z kropką jako separatorem dziesiętnym), oczekiwanego przez API.

Zdj. 9 Pogląd na metodę obsługi zapytań API

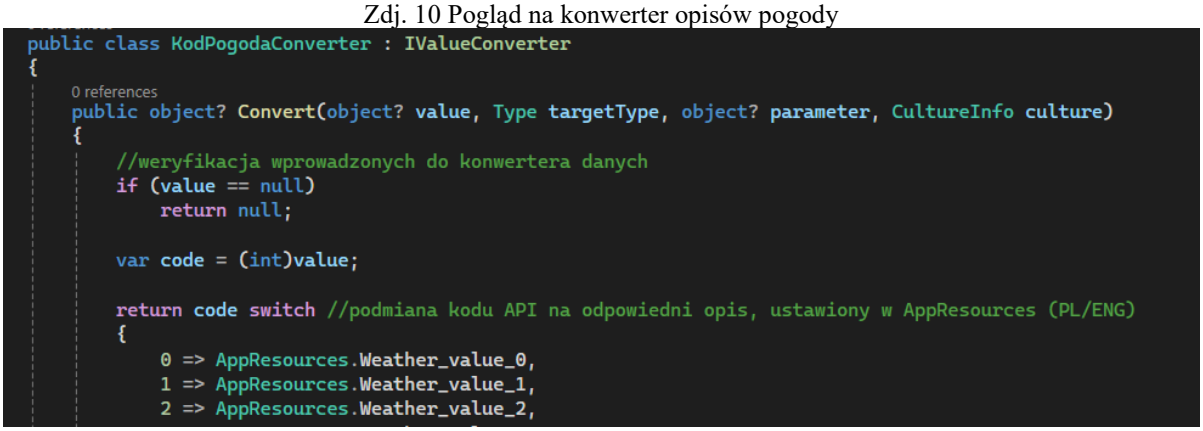
```
//Pobieranie informacji z API
1 reference
private async Task GetWeather(Location location)
{
    string latitude = location.Latitude.ToString(CultureInfo.InvariantCulture); //konwersja formatu liczbowego PL na EN
    string longitude = location.Longitude.ToString(CultureInfo.InvariantCulture); //konwersja formatu liczbowego PL na EN

    var url = $"https://api.open-meteo.com/v1/forecast?latitude={latitude}&longitude={longitude}&daily=weather_code,apparent_t
    IsLoading = true; //widoczność ActivityIndicator
    var response = await HttpClient.GetAsync(url);

    if (response.IsSuccessStatusCode)
    {
        using (var responseStream = await response.Content.ReadAsStreamAsync())
        {
            var data = await JsonSerializer.DeserializeAsync<PogodaData>(responseStream);
            PogodaData = data;

            for (int i = 1; i < PogodaData.daily.time.Length-1; i++) //pobieranie danych dla każdego kolejnego dnia z osobna (
            {
                var daily_single = new Daily_Single...;
```

Źródło: Praca własna w Visual Studio 2022

2.4. Klasa *KodPogodaConverter* - przypisanie opisu pogody

Klasa *KodPogodaConverter* implementuje interfejs *IValueConverter* i wykorzystuje konstrukcję *Switch Expression* do mapowania kodów pogodowych z API na odpowiednie opisy w aplikacji. Dla każdego kodu z API przypisana jest odpowiednia wartość tekstowa, która jest dostosowana do języka systemu urządzenia użytkownika, dzięki wykorzystaniu zasobów aplikacji AppResources.

Zdj. 10 Pogląd na konwerter opisów pogody



```
public class KodPogodaConverter : IValueConverter
{
    0 references
    public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
    {
        //weryfikacja wprowadzonych do konwertera danych
        if (value == null)
            return null;

        var code = (int)value;

        return code switch //podmiana kodu API na odpowiedni opis, ustawiony w AppResources (PL/ENG)
        {
            0 => AppResources.Weather_value_0,
            1 => AppResources.Weather_value_1,
            2 => AppResources.Weather_value_2,
```

Źródło: Praca własna w Visual Studio 2022

2.5. Klasa *KodGrafikaConverter* - przypisanie grafiki

Klasa *KodGrafikaConverter* działa podobnie do poprzedniego konwertera, implementując interfejs *IMultiValueConverter* i stosując konstrukcję Switch Expression do przypisania odpowiednich grafik pogodowych na podstawie kodów pogodowych z API oraz pory dnia. Dodatkowo, utworzono tablice, które zawierają konkretne kody pogodowe przypisane do danego rodzaju grafiki, ponieważ kilka różnych kodów może odpowiadać jednej grafice.

Program sprawdza, czy w danej lokalizacji jest dzień, czy noc, i na tej podstawie dostosowuje grafikę pogodową. Zastosowanie *MultiBinding* pozwala na jednoczesne wiązanie wielu wartości (kodu pogodowego i kodu dnia), co umożliwia dynamiczną zmianę animacji w zależności od stanu pogody i pory dnia. Dla prognozy pogody w kolejnych dniach domyślnie przypisywana jest grafika związana z dniem.

Zdj. 11 Pogląd na konwerter grafik pogody

```
public class KodGrafikaConverter : IMultiValueConverter
{
    0 references
    public object? Convert(object[] values, Type targetType, object? parameter, CultureInfo culture)
    {
        //weryfikacja wprowadzonych do konwertera danych - musi pobrac najpierw kod API, potem kod dnia
        if (values.Length < 2 || values[0] is not int code || values[1] is not int isDay)
            return null;

        var lottieImageSource = new SKFileLottieImageSource();

        //przypisanie kodów API do rodzaju grafiki (po kilka na jeden dla uproszczenia)
        var rainCodes = new[] { 51, 53, 55, 61, 63, 65 };
        var rainSnowCodes = new[] { 56, 57 };
        var snowCodes = new[] { 56, 57, 66, 67, 71, 73, 75, 77, 85, 86 };
        var thunderCodes = new[] { 95, 96, 99 };
        var showerCodes = new[] { 80, 81, 82 };
        var fogCodes = new[] { 45, 48 };

        string file = code switch //podmiana kodu API na odpowiednią grafikę
        {
            0 or 1 => isDay == 1 ? "dayClear.json" : "nightClear.json",
            2 => isDay == 1 ? "dayCloudPart.json" : "nightCloudPart.json",
            3 => isDay == 1 ? "dayCloud.json" : "nightCloud.json",
            _ when fogCodes.Contains(code) => isDay == 1 ? "dayFog.json" : "nightFog.json",
            _ when rainCodes.Contains(code) => isDay == 1 ? "dayRain.json" : "nightRain.json",
            _ when rainSnowCodes.Contains(code) => isDay == 1 ? "dayRainSnow.json" : "nightRainSnow.json",
            _ when snowCodes.Contains(code) => isDay == 1 ? "daySnow.json" : "nightSnow.json",
            _ when showerCodes.Contains(code) => isDay == 1 ? "dayRainShower.json" : "nightRainShowerw.json",
            _ when thunderCodes.Contains(code) => isDay == 1 ? "dayThunder.json" : "nightThunder.json",
            _ => "dayClear.json"
        };

        lottieImageSource.File = file; //przypisanie grafiki
        return lottieImageSource;
```

Źródło: Praca własna w Visual Studio 2022

3. Style i zasoby aplikacji

Aplikacja posiada zdefiniowane style, kolory, tłumaczenie i własną ikonę oraz ekran ładowania. W zasobach znajdują się również grafiki prezentujące pogodę oraz ikony gps.

3.1. Identyfikacja wizualna

Kolorystykę i style aplikacji scharakteryzowano w Resources/Styles/Colors.xaml i Resources/Styles/Styles.xaml). Pozwala to zachować spójność interfejsu użytkownika i łatwo wprowadzać modyfikacje elementów.

Zdj. 12 Schemat pliku Colors

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

    <!-- Note: For Android please see also Platforms\Android\Resources\values\colors.xml -->
    <Color x:Key="Primary">#F0F2FA</Color>
    <Color x:Key="PrimaryText">#222831</Color>
    <Color x:Key="PrimaryDark">#191D24</Color>
    <Color x:Key="PrimaryDarkText">#FFFDF6</Color>
    <Color x:Key="Secondary">#E0E4F5</Color>
    <Color x:Key="SecondaryText">#404040</Color>
    <Color x:Key="SecondaryDark">#222831</Color>
    <Color x:Key="SecondaryDarkText">#ADADAD</Color>

    <Color x:Key="White">White</Color>
    <Color x:Key="Black">Black</Color>
```

Źródło: Praca własna w Visual Studio 2022

Zdj. 13 Schemat pliku Styles

```
    General

    Main info

    <!--#region Collection View -->
    <Style TargetType="VerticalStackLayout" x:Key="CollectionView"...>

    <Style TargetType="Border" x:Key="CollectionViewContainer"...>

    <Style TargetType="Label" x:Key="CollectionViewItem"...>

    <Style TargetType="Label" x:Key="CollectionViewItem_TempMax"...>

    <Style TargetType="Label" x:Key="CollectionViewItem_TempMin"...>

    <Style TargetType="Label" x:Key="CollectionViewItem_Title"...>

    <Style TargetType="Label" x:Key="CollectionViewItem_Header">
        <Setter Property="FontSize" Value="15"/>
        <Setter Property="FontAttributes" Value="Bold"/>
        <Setter Property="HorizontalOptions" Value="Center"/>
        <Setter Property="TextColor" Value="{AppThemeBinding Light={StaticResource PrimaryText}, Dark={StaticResource PrimaryDarkText}}"/>
    </Style>
    <!--#endregion-->
</ResourceDictionary>
```

Źródło: Praca własna w Visual Studio 2022

Ikona domyślna została podmieniona własną ilustracją w formacie .svg (plik Resources/AppIcon/appicon.svg), podobnie z ekranem ładowania (plik Resources/Splash/splash.svg).

3.2. Grafiki i animacje

Ikona przycisku wyszukania lokalizacji posiada dwie wersje – dla trybu jasnego i dla trybu ciemnego. Grafiki .png zostały zamieszczone została w ścieżce *Resources/Images*.

Animacje pogody w formacie *json*. umieszczono w ścieżce *Resources/Raw* Do obsługi animacji niezbędne było zainstalowanie rozszerzenia *SkiaSharp.Extended.UI.*Maui, które wprowadza nowy widok *SKLottieView*.

Zdj. 14 Implemnetacja grafiki json

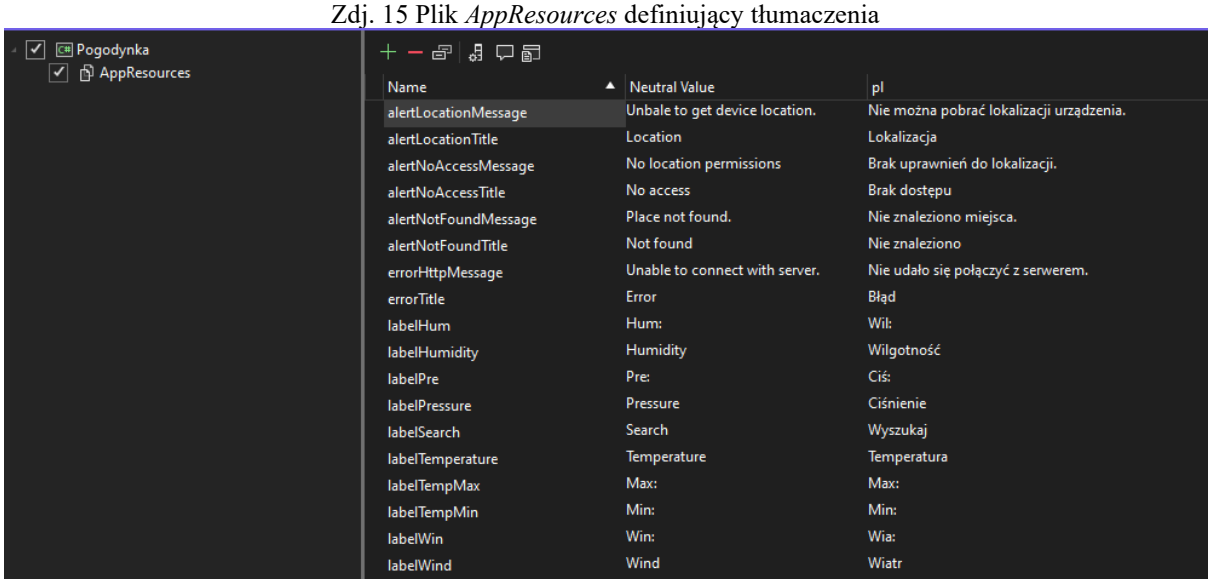
```
<!-- Grafika -->
<SkiaSharp:SKLottieView  Grid.Row="1" Grid.ColumnSpan="2" RepeatCount="-1" HeightRequest="264">
    <SkiaSharp:SKLottieView.Source>
        <MultiBinding Converter="{StaticResource KodGrafikaConverter}">
            <Binding Path="PogodaData.current.weather_code"/>
            <Binding Path="PogodaData.current.is_day"/>
        </MultiBinding>
    </SkiaSharp:SKLottieView.Source>
</SkiaSharp:SKLottieView>
```

Źródło: Praca własna w Visual Studio 2022

3.3. Dwujęzyczność aplikacji

Aplikacja posiada dwie wersje językowe: angielską i polską – zależne od ustawień systemu. W przypadku ustawienia języka innego niż polski i angielski, stosowana jest angielska wersja językowa. Wersje językowe zaimplementowano w folderze Resources/Strings, jako neutralny język wybrano English (United States) – en-US (*Project properties -> Package -> General -> Assembly neutral language*).

Zdj. 15 Plik *AppResources* definiujący tłumaczenia



Name	Neutral Value	pl
alertLocationMessage	Unbale to get device location.	Nie można pobrać lokalizacji urządzenia.
alertLocationTitle	Location	Lokalizacja
alertNoAccessMessage	No location permissions	Brak uprawnień do lokalizacji.
alertNoAccessTitle	No access	Brak dostępu
alertNotFoundMessage	Place not found.	Nie znaleziono miejsca.
alertNotFoundTitle	Not found	Nie znaleziono
errorHttpMessage	Unable to connect with server.	Nie udało się połączyć z serwerem.
errorTitle	Error	Błąd
labelHum	Hum:	Wil:
labelHumidity	Humidity	Wilgotność
labelPre	Pre:	Ciś:
labelPressure	Pressure	Ciśnienie
labelSearch	Search	Wyszukaj
labelTemperature	Temperature	Temperatura
labelTempMax	Max:	Max:
labelTempMin	Min:	Min:
labelWin	Win:	Wia:
labelWind	Wind	Wiatr

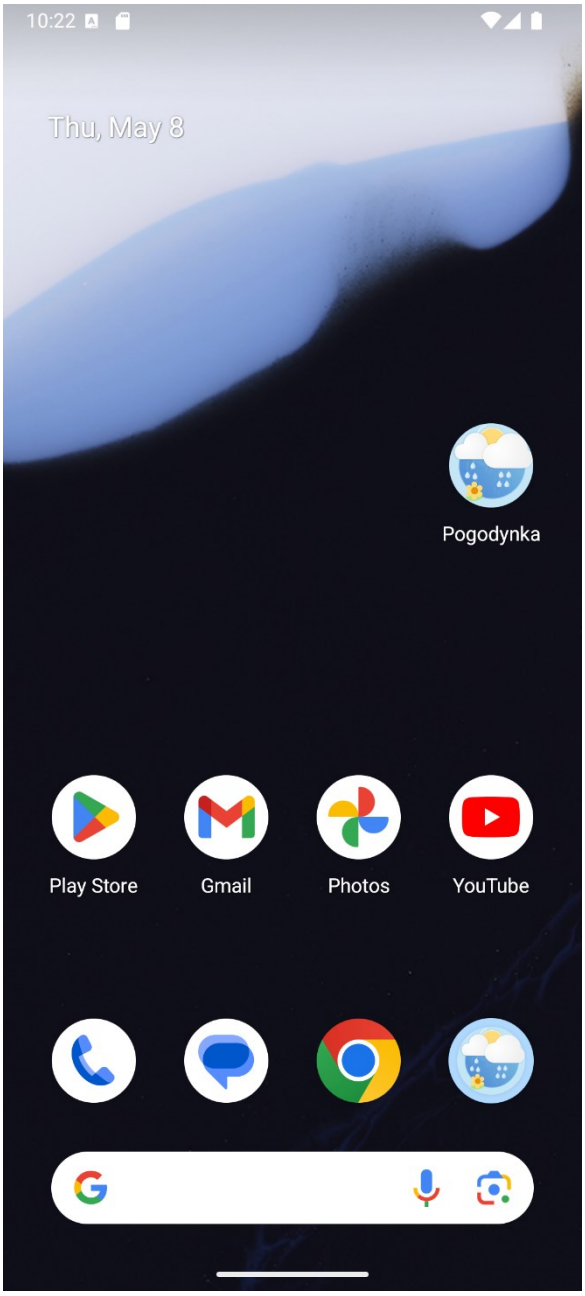
Źródło: Praca własna w programie Visual Studio 2022

4. Prezentacja aplikacji

4.1. Ikona i ekran ładowania aplikacji

Aplikacja posiada własną ikonę i ekran ładowania aplikacji, prezentujący logo.

Zdj. 16 Podgląd ikony na ekranie telefonu



Źródło: Praca własna

Zdj. 17 Podgląd ekranu ładowania aplikacji

Źródło: Praca własna

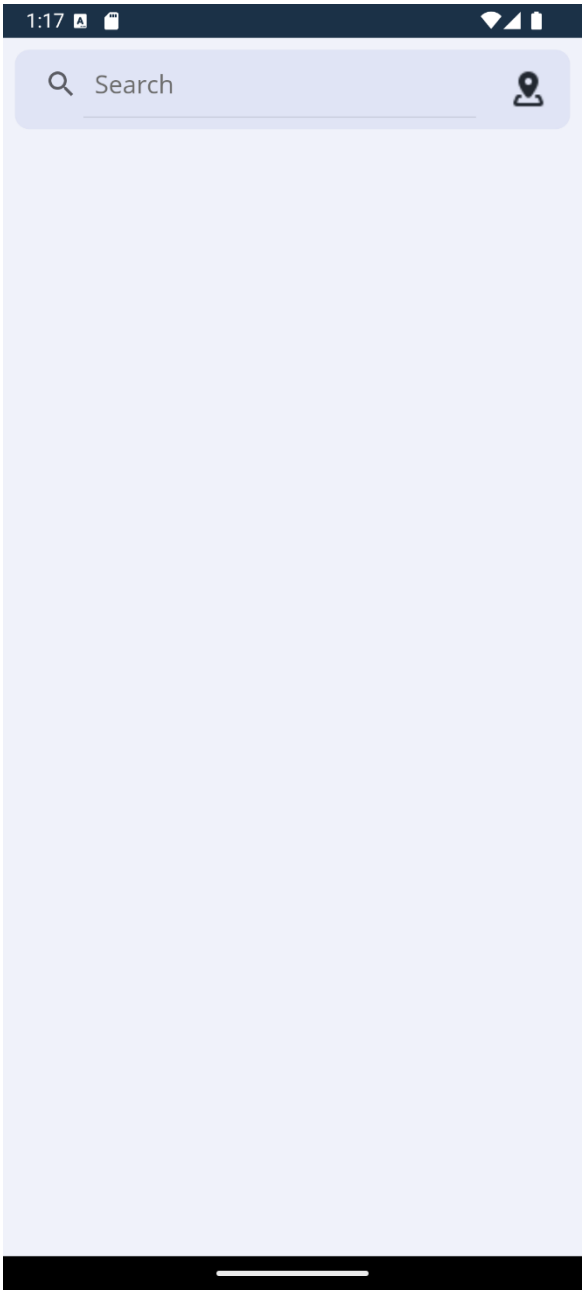
4.2. Strona początkowa i wyszukiwanie miejscowości

Po uruchomieniu aplikacji, użytkownik widzi tylko pasek wyszukiwania. Główna zawartość aplikacji pojawia się dopiero po pierwszym wyszukaniu.

Aby wyświetlić informacje o pogodzie, użytkownik musi wyszukać jakąś lokalizację lub użyć opcji automatycznego wykrycia lokalizacji.

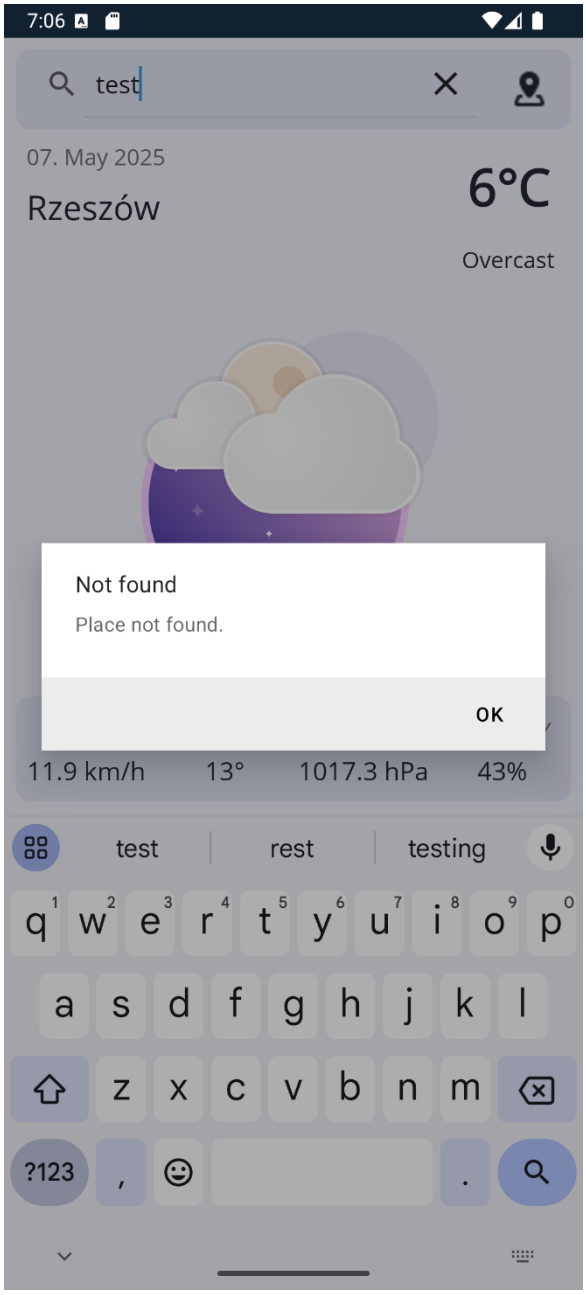
W przypadku, gdy miejscowość nie została znaleziona, użytkownik otrzyma powiadomienie. Podczas procesu wyszukiwania użytkownik widzi animację procesowania.

Zdj. 18 Podgląd ekranu pierwszego uruchomienia



Źródło: Praca własna

Zdj. 19 Podgląd błędu wyszukiwania



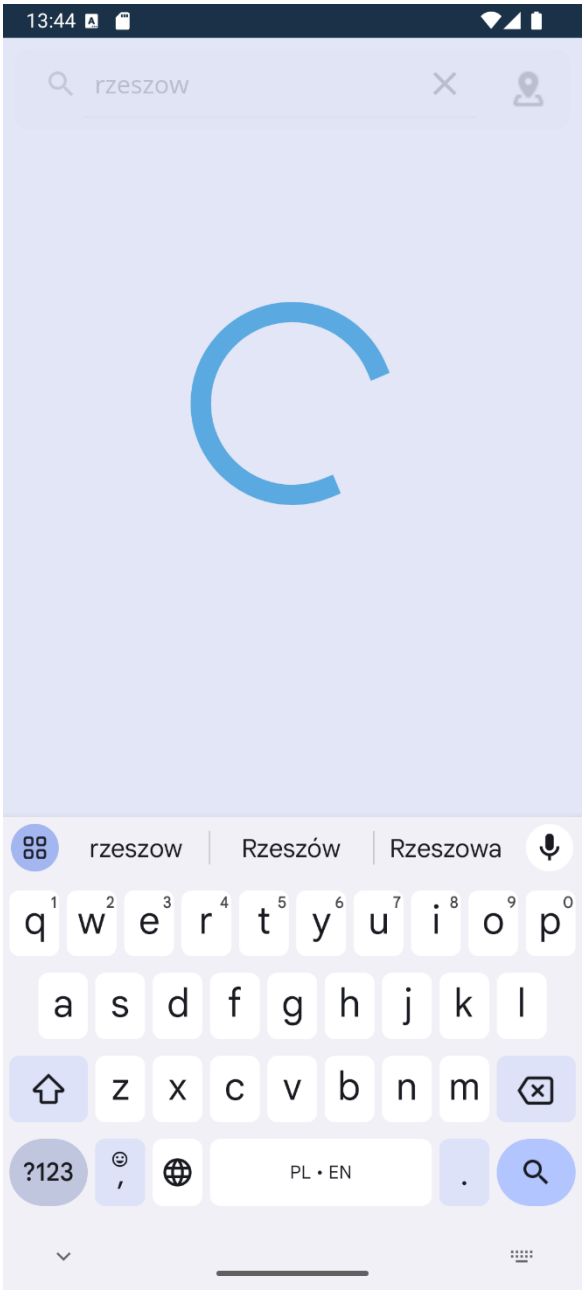
Źródło: Praca własna

4.3. Strona główna

Strona główna aplikacji zawiera informacje o pogodzie w danej miejscowości. W nagłówku strony widnieje pasek wyszukiwania, a pod nim znajdują się: dzisiejsza data, wybrana miejscowość i aktualna temperatura (odczuwalna).

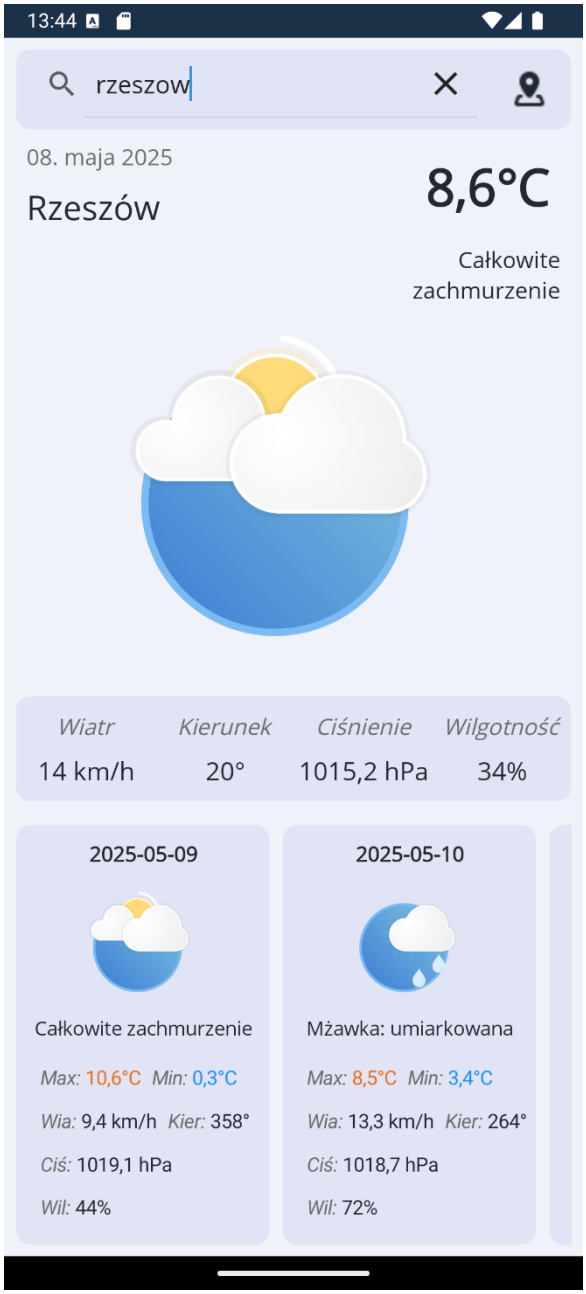
Pod nagłówkiem wstawiono graficzną reprezentację bieżącej pogody wraz ze szczegółami: opis pogody, prędkość i kierunek wiatru, ciśnienie i wilgotność. Ostatnia sekcja zawiera prognozę na kolejnych 5 dni: data, grafika, temperatura maksymalna i minimalna, prędkość wiatru, kierunek wiatru, ciśnienie oraz wilgotność.

Zdj. 20 Podgląd procesu wyszukiwania



Źródło: Praca własna

Zdj. 21 Podgląd ekranu głównego

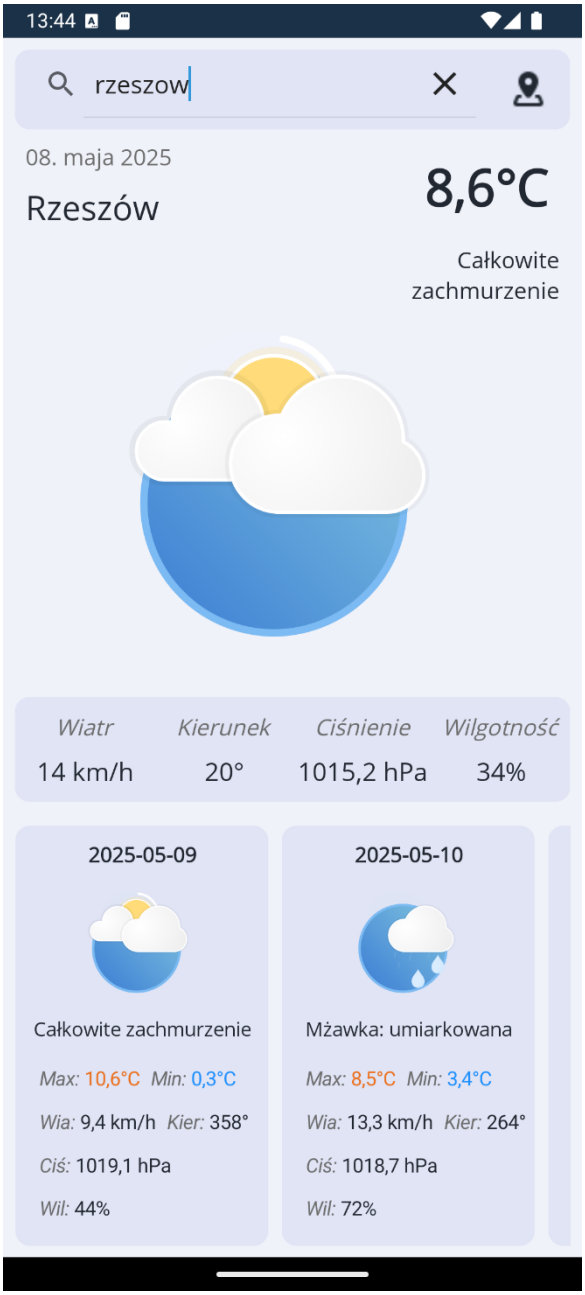


Źródło: Praca własna

4.4. Tryb jasny i ciemny

Aplikacja dostosowuje swoją kolorystykę do ustawień systemowych telefonu. Tryb jasny (w odcieniach bieli) jest zastosowany dla jasnego motywu, tryb ciemny (w odcieniach czerni) jest zastosowany dla motywu ciemnego.

Zdj. 22 Podgląd trybu jasnego



Źródło: Praca własna

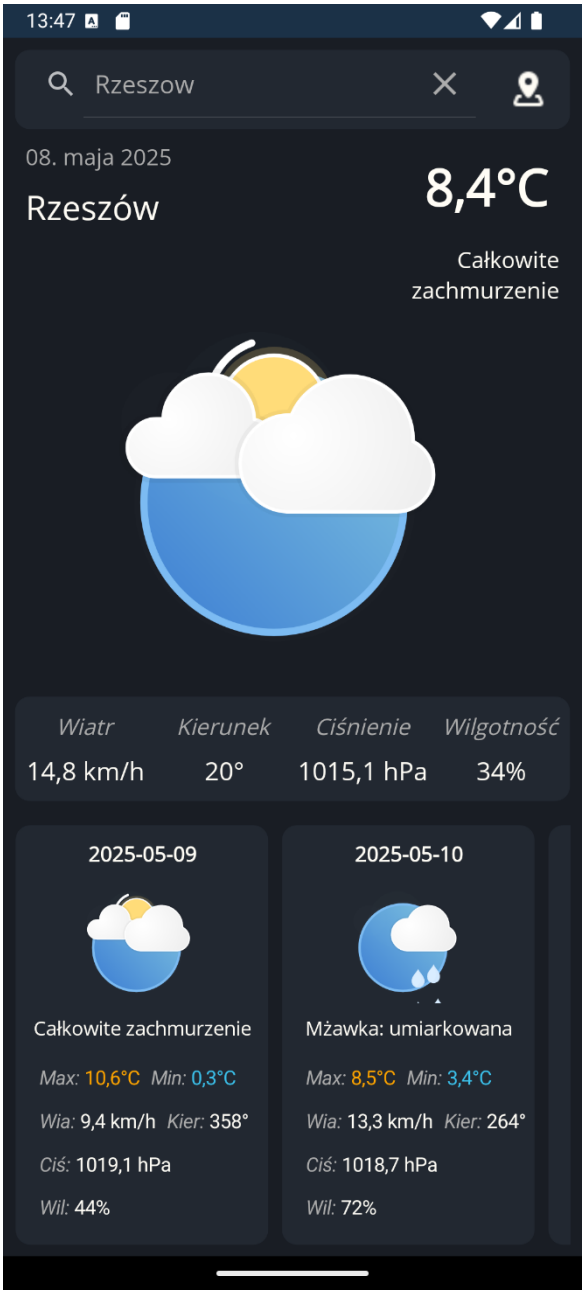
Zdj. 23 Podgląd trybu ciemnego

Źródło: Praca własna

4.5. Dwujęzyczność

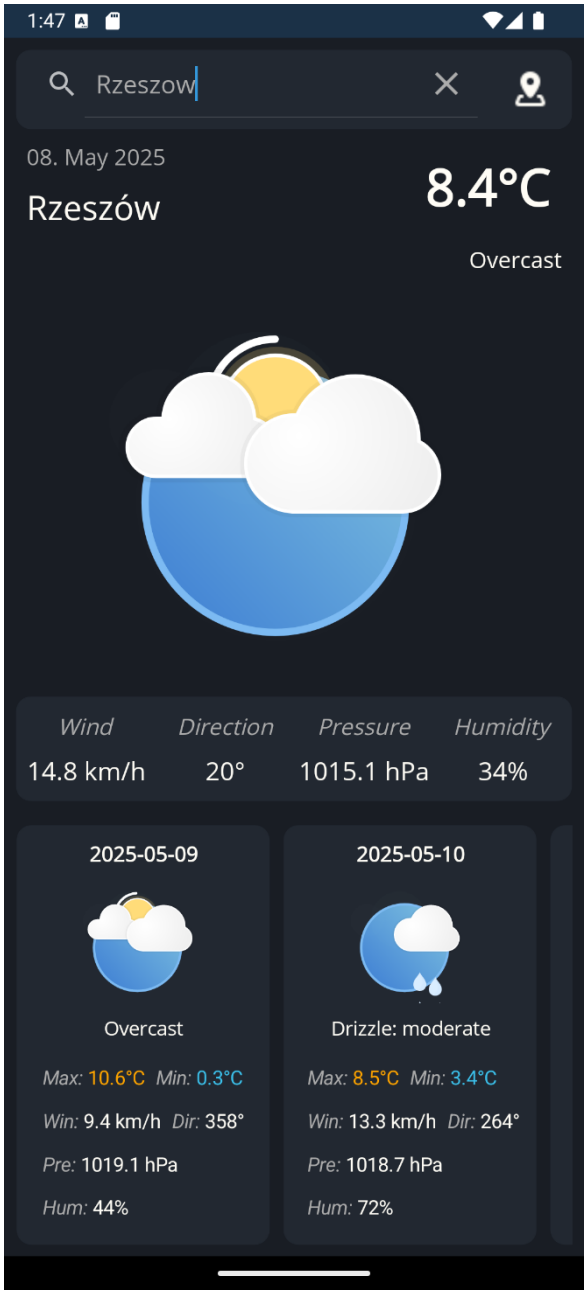
Aplikacja dostosowuje język wyświetlania do ustawień systemowych telefonu. Jeśli językiem systemowym jest język polski, to aplikacja wyświetla informacje po Polsku. W innym wypadku stosowany jest język angielski (jako uniwersalny).

Zdj. 24 Podgląd języka polskiego



Źródło: Praca własna

Zdj. 25 Podgląd języka angielskiego



Źródło: Praca własna

5. Podsumowanie projektu

Repozytorium

Projekt wyeksportowano za pomocą Git. Kod aplikacji znajduje się pod linkiem: https://github.com/iguanaiza/Pogodynka

Podsumowanie

Stworzono przydatną aplikację i zaimplementowano wszystkie wymagane funkcjonalności. Pomimo napotkanych problemów z konwersją danych liczbowych, udało się zaimplementować skuteczne rozwiązanie.

Aplikację można w przyszłości rozbudować dodając dodatkowe ustawienia użytkownika (przełączanie języka, motywu z poziomu aplikacji) oraz obsługę podpowiedzi lokalizacji w wyszukiwarce.

6. Netografia

1. *Słownik dla mediów. Najważniejsze pojęcia i zwroty w meteorologii*. METEO IMGW-PIB https://imgw.pl/wp-content/uploads/2024/10/slownik-dla-mediow_cmok_0.pdf
2. *Weather Forecast API docs.* (2024). Open-Meteo. https://open-meteo.com/en/docs
3. *PropertyChanged.Fody.* (2025). Fody. https://github.com/Fody/PropertyChanged
4. *Reuse SkiaSharp code in .NET MAUI.* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/migration/skiasharp?view=net-maui-9.0
5. *.NET MAUI 9: Geolocation.* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/device/geolocation
6. *.NET MAUI 9: Commanding.* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/commanding?view=net-maui-9.0
7. *.NET MAUI 9: Localization.* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/localization?view=net-maui-9.0
8. *C#: Switch expression - pattern matching expressions using the switch keyword.* (2022). Microsoft. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/switch-expression
9. *.NET MAUI 9: Data binding.* (2025). Microsoft. https://learn.microsoft.com/en-gb/dotnet/maui/fundamentals/data-binding/?view=net-maui-9.0
10. *.NET MAUI 9: Binding mode.* (2025). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/binding-mode?view=net-maui-9.0
11. *.NET MAUI 9: ObservableCollection<T> Class.* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection-1?view=net-9.0
12. *.NET MAUI 9: Add images to a .NET MAUI app project.* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/user-interface/images/images?view=net-maui-9.0
13. *.NET MAUI 9: Create resource files for .NET apps* (2023). Microsoft. https://learn.microsoft.com/en-us/dotnet/core/extensions/create-resource-files
14. *.NET MAUI 9: Add a splash screen to a .NET MAUI app project* (2024). Microsoft. https://learn.microsoft.com/en-us/dotnet/maui/user-interface/images/splashscreen?view=net-maui-9.0&tabs=android