Dokumentacja Techniczna Aplikacji Pogodowej WeatherAPIApp

(Dariusz Salzburg)

Spis Treści

---

1. Wprowadzenie

Aplikacja pogodowa została stworzona z wykorzystaniem frameworka .NET MAUI, umożliwiającego tworzenie aplikacji wieloplatformowych. Głównym celem aplikacji jest dostarczanie użytkownikowi aktualnych informacji pogodowych, prognoz oraz rekomendacji opartych na preferencjach użytkownika i aktualnej pogodzie. Aplikacja pogodowa pozwala użytkownikowi na wyszukiwanie prognozy pogody w wybranych lokalizacjach. Użytkownik może przeglądać zarówno aktualne warunki pogodowe, jak i prognozy na kilka dni w przyszłość. Aplikacja korzysta z danych z zewnętrznego API pogodowego, wyświetlając takie informacje jak temperatura, wilgotność, ciśnienie oraz wschód i zachód słońca.

2. Architektura Aplikacji

2.1. Struktura Projektu

Aplikacja jest zorganizowana zgodnie z wzorcem projektowym MVVM (Model-View-ViewModel), co ułatwia zarządzanie kodem oraz separację logiki biznesowej od interfejsu użytkownika.

- **ViewModel-e**: Zarządzają logiką biznesową oraz danymi prezentowanymi w widokach.
- **Modele**: Reprezentują dane oraz logikę związaną z przetwarzaniem danych.
- **Usługi**: Dostarczają funkcjonalności niezwiązane bezpośrednio z logiką biznesową, takie jak pobieranie lokalizacji.
- **Widoki (Views)**: Definiują interfejs użytkownika za pomocą XAML.

2.2. Wykorzystane Technologie i Biblioteki

- **.NET MAUI**: Framework do tworzenia aplikacji wieloplatformowych.
- **WeatherAPI**: Zewnętrzne API do pobierania danych pogodowych.
- **MVVM Helpers**: Ułatwienia dla implementacji wzorca MVVM.
- **Newtonsoft.Json**: Biblioteka do serializacji i deserializacji JSON.

---

3. Opis Klas i Komponentów

3.1. ViewModel-e

3.1.1. `WeatherPageViewModel`

Opis: `WeatherPageViewModel` zarządza logiką biznesową dla strony głównej aplikacji pogodowej.

Kluczowe Właściwości:

- `LocationName`: Nazwa aktualnej lokalizacji.
- `CurrentWeather`: Obiekt zawierający aktualne dane pogodowe.
- `WeatherForecastHours`: Prognoza godzinowa (kolekcja obiektów `Hour`).
- `WeatherForecastDays`: Prognoza dzienna (kolekcja obiektów `Forecastday`).
- `PreferenceManager`: Obiekt zarządzający preferencjami użytkownika i generujący rekomendacje.

Kluczowe Metody:

- `InitializeLocation()`: Inicjalizuje lokalizację użytkownika.
- `GetWeatherData()`: Pobiera dane pogodowe dla aktualnej lokalizacji.
- `RefreshWeatherData()`: Odświeża dane pogodowe.
- `ApplyQueryAttributes(IDictionary<string, object> query)`: Obsługuje parametry nawigacji.

Przykład Implementacji Metody `GetWeatherData()`:

```
public async Task GetWeatherData()
{
    try
    {
        WeatherAPIClient client = new();
        var forecastWeather = await
client.APIs.GetForecastWeatherAsync(LocationName, 5);

        LocationName = forecastWeather.Location.Name;
        CurrentWeather = forecastWeather.Current;

        var weatherForecastDaysClone = new
List<Forecastday>(forecastWeather.Forecast.Forecastday);
        WeatherForecastDays = new
ObservableCollection<Forecastday>(weatherForecastDaysClone);

        PreferenceManager.CurrentWeather = forecastWeather.Current;

        // Pobieranie prognozy godzinowej na najbliższe 48 godzin
        var weatherForecastHours48 = new
List<Hour>(forecastWeather.Forecast.Forecastday[0].Hours);
        weatherForecastHours48.AddRange(new
List<Hour>(forecastWeather.Forecast.Forecastday[1].Hours));

        WeatherForecastHours = new
ObservableCollection<Hour>(weatherForecastHours48);
    }
    catch (Exception ex)
    {
        // Obsługa błędów
        Application.Current.Dispatcher.Dispatch(() =>
        {
            Application.Current.MainPage.DisplayAlert("WeatherApp", ex.Message,
"OK");
        });
    }
}
```

3.1.2. `PageViewModel`

Opis: `PageViewModel` obsługuje logikę dla stron takich jak `RainPage`, `SunPage` i `TempPage i WindPage`

Kluczowe Właściwości:

- `WeatherForcastDays`: Kolekcja prognoz pogodowych dla różnych lokalizacji.
- `SelectLocationCommand`: Komenda pozwalająca na wybór konkretnej lokalizacji.
- `SearchLocationCommand`: Komenda do wyszukiwania i dodawania nowej lokalizacji.

Kluczowe Metody:

- `GetFavoriteData()`: Pobiera dane pogodowe dla ulubionych lokalizacji.
- `SearchAndUpdateLocation(string newLocationName)`: Wyszukuje nową lokalizację i aktualizuje dane.

Przykład Implementacji Metody `SearchAndUpdateLocation(string newLocationName)`:

```
public async Task SearchAndUpdateLocation(string newLocationName)
{
    try
    {
        WeatherAPIClient weatherAPIClient = new();
        var forecastWeather = await
weatherAPIClient.APIs.GetForecastWeatherAsync(newLocationName, 5);

        if (forecastWeather != null)
        {
            bool locationExists = locationNames.Contains(newLocationName);
            if (!locationExists)
            {
                locationNames.Add(newLocationName);
                SaveLocations();
                WeatherForcastDays.Clear();
                await GetFavoriteData();
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error fetching weather data: {ex.Message}");
    }
}
```

3.2. Modele

3.2.1. `PreferenceManager`

Opis: Klasa `PreferenceManager` zarządza preferencjami użytkownika i generuje rekomendacje na podstawie aktualnej pogody.

Kluczowe Właściwości:

- `SelectedPreference`: Aktualnie wybrana preferencja użytkownika (np. "Indoor Activities").
- `Preferences`: Lista dostępnych preferencji.
- `Recommendation`: Generowana rekomendacja oparta na wybranej preferencji i aktualnej pogodzie.
- `CurrentWeather`: Aktualne dane pogodowe.

Kluczowe Metody:

- `GenerateRecommendationBasedOnPreference()`: Generuje rekomendację po zmianie preferencji lub aktualizacji danych pogodowych.

Przykład Generowania Rekomendacji:

```
public void GenerateRecommendationBasedOnPreference()
{
    if (CurrentWeather == null) return;

    double tempC = (double)currentWeather.TempC;
    string condition = currentWeather.Condition.Text.ToLower();

    if (SelectedPreference == "Indoor Activities")
    {
        if (condition.Contains("rain") || condition.Contains("snow"))
        {
            Recommendation = "It looks like a great day to stay indoors and
enjoy a book or movie.";
        }
        else
        {
            Recommendation = "Perfect weather for indoor activities, but you can
also enjoy the outdoors!";
        }
    }
    // ... pozostałe warunki
}
```

3.3. Usługi

3.3.1. `LocationService`

Opis: `LocationService` odpowiada za pobieranie aktualnej lokalizacji użytkownika.

Kluczowe Metody:

- `GetLocationAsync()`: Pobiera aktualną lokalizację geograficzną użytkownika.
- `GetLocationNameAsync(double latitude, double longitude)`: Zwraca nazwę lokalizacji na podstawie współrzędnych.

Przykład Implementacji Metody `GetLocationAsync()`:

```
public async Task<(string locationName, string locationCoordinates, string
alert)> GetLocationAsync()
{
    try
    {
        var location = await Geolocation.GetLastKnownLocationAsync() ??
                       await Geolocation.GetLocationAsync(new
GeolocationRequest(GeolocationAccuracy.High));

        if (location != null)
        {
            double roundedLatitude = Math.Round(location.Latitude, 2);
            double roundedLongitude = Math.Round(location.Longitude, 2);

            string locationName = await GetLocationNameAsync(roundedLatitude,
roundedLongitude);
            string locationCoordinates = $"{roundedLatitude},
{roundedLongitude}";

            return (locationName, locationCoordinates, null);
        }
```

```
        else
        {
            return ("Warszawa", string.Empty, "Nie można uzyskać lokalizacji");
        }
    }
    catch (Exception)
    {
        return ("Warszawa", string.Empty, "Błąd podczas pobierania
lokalizacji.");
    }
}
```

3.4.Konwertery

Konwertery w aplikacji służą do przekształcania danych wyświetlanych w interfejsie użytkownika. Są szczególnie przydatne w kontekście formatowania wartości pobieranych z API pogodowego, takich jak daty, czas oraz prędkości.

3.4.1. `DateToDayOfWeekConverter`

Opis:
Ten konwerter służy do przekształcania obiektu daty na dzień tygodnia. Przykładowo, konwertuje datę w formacie `DateTime` na nazwę dnia tygodnia (np. "Monday", "Tuesday").

Kluczowe Właściwości:

- Oczekuje obiektu typu `DateTime` lub ciągu znaków, który może być sparsowany na datę.
- Zwraca nazwę dnia tygodnia w formie łańcucha znaków.

Przykład Użycia:

```
//XAML
<Label Text="{Binding Date, Converter={StaticResource
DateToDayOfWeekConverter}}" />


public class DateToDayOfWeekConverter : IValueConverter
{
    public string? Convert(object? value, Type targetType, object? parameter,
CultureInfo culture)
    {
        if (value == null) return null;

        if (DateTime.TryParse(value.ToString(), out DateTime dateTime))
        {
            return dateTime.DayOfWeek.ToString();
        }

        return null;
    }

    public object? ConvertBack(object? value, Type targetType, object?
parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

3.4.2. `KmHToMsConverter`

Opis:
Konwerter ten przekształca prędkość z jednostek kilometrów na godzinę (km/h) na metry na sekundę (m/s). Jest to szczególnie przydatne przy prezentacji danych meteorologicznych dotyczących prędkości wiatru.

Kluczowe Właściwości:

- Przyjmuje wartość typu `double` (km/h).
- Zwraca wartość w formacie m/s, z zaokrągleniem do jednego miejsca po przecinku.

Przykład Użycia:

```
<Label Text="{Binding WindKph, Converter={StaticResource KmHToMsConverter}}" />

public class KmHToMsConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        if (value is double kmhValue)
        {
            double msValue = kmhValue * 0.27778;
            return msValue.ToString("F1");
        }

        return value;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

3.4.3. `TimeFormatterConverter`

Opis:
Konwerter ten formatuje ciąg znaków reprezentujący czas (np. "2024-09-12 15:00") na format godziny w formacie 24-godzinnym ("HH", np. "15:00").
Kluczowe Właściwości:

- Przyjmuje wartość typu `string`, która reprezentuje datę i godzinę.
- Zwraca sformatowany czas jako ciąg znaków w formacie ("HH", np. "15:00").

Przykład Użycia:

```
<Label Text="{Binding TimeString, Converter={StaticResource
TimeFormatterConverter}}" />

public class TimeFormatterConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
    {
```

```
        if (value is string timeString)
        {
            if (DateTime.TryParse(timeString, out DateTime dateTime))
            {
                return dateTime.ToString("HH:mm");
            }
        }
        return value;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

---

4. Strony i Interfejs Użytkownika

4.1. `WeatherPage`

Opis: Główna strona aplikacji, prezentująca aktualne dane pogodowe, prognozy oraz rekomendacje.

Kluczowe Elementy UI:

- **Entry do wprowadzania lokalizacji:**

  ```
  <Entry
      Placeholder="Enter location"
      Text="{Binding UserInputLocation}"
      Completed="OnLocationEntryCompleted" />
  ```

- **Wyświetlanie aktualnej temperatury:**

  ```
  <HorizontalStackLayout Margin="0,10,0,0" HorizontalOptions="Center">
      <Label FontSize="45" Text="{Binding CurrentWeather.TempC}" />
      <Label FontSize="20" Text="o" />
      <Label FontSize="45" Text="C" />
  </HorizontalStackLayout>
  ```

- **Picker do wyboru preferencji:**

  ```
  <Picker
      ItemsSource="{Binding PreferenceManager.Preferences}"
      SelectedItem="{Binding PreferenceManager.SelectedPreference}"
      Title="Choose preference"
      HorizontalOptions="Center"/>
  ```

- **Wyświetlanie rekomendacji:**

  ```
  <Frame BackgroundColor="White" CornerRadius="10" Padding="15">
      <Label Text="{Binding PreferenceManager.Recommendation}" FontSize="18"
  FontAttributes="Bold" TextColor="Black" />
  </Frame>
  ```

- **Prognoza godzinowa:**

```
<CollectionView ItemsSource="{Binding WeatherForecastHours}">
    <!-- Template dla elementów -->
</CollectionView>
```

4.2. `RainPage`

Opis: Strona prezentująca informacje o opadach oraz wilgotności dla wybranych lokalizacji.

Kluczowe Elementy UI:

- **SearchBar do wyszukiwania miast:**

```
<SearchBar
    Placeholder="Search for a city"
    SearchCommand="{Binding SearchLocationCommand}"
    SearchCommandParameter="{Binding Text, Source={RelativeSource
Self}}" />
```

- **Lista prognoz pogodowych:**

```
<CollectionView ItemsSource="{Binding WeatherForcastDays}">
    <!-- Template dla elementów -->
</CollectionView>
```

- **Wyświetlanie wilgotności i opadów:**

```
<VerticalStackLayout Grid.Column="1">
    <Label Text="Humidity" />
    <Label Text="{Binding Current.Humidity}" />
</VerticalStackLayout>
<VerticalStackLayout Grid.Column="3">
    <Label Text="Rain" />
    <Label Text="{Binding Current.PrecipMm}" />
</VerticalStackLayout>
```

4.3. `SunPage`

Opis: Strona skupiająca się na indeksie UV oraz stopniu zachmurzenia.

Kluczowe Elementy UI:

- **Wyświetlanie indeksu UV i zachmurzenia:**

```
<VerticalStackLayout Grid.Column="1">
    <Label Text="Index UV" />
    <Label Text="{Binding Current.Uv}" />
</VerticalStackLayout>
<VerticalStackLayout Grid.Column="3">
    <Label Text="Cloud (%)" />
    <Label Text="{Binding Current.Cloud}" />
</VerticalStackLayout>
```

- **Prognoza godzinowa indeksu UV i zachmurzenia:**

```
<CollectionView ItemsSource="{Binding ForecastHours}">
    <!-- Template dla elementów -->
</CollectionView>
```

4.4. `TempPage`

Opis: Strona prezentująca aktualną temperaturę oraz temperaturę odczuwalną.

Kluczowe Elementy UI:

- **Wyświetlanie temperatur:**

```
<VerticalStackLayout Grid.Column="1">
    <Label Text="Temp" />
    <Label Text="{Binding Current.TempC}" />
</VerticalStackLayout>
<VerticalStackLayout Grid.Column="3">
    <Label Text="Feel Temp" />
    <Label Text="{Binding Current.FeelslikeC}" />
</VerticalStackLayout>
```

- **Prognoza godzinowa temperatur:**

```
<CollectionView ItemsSource="{Binding ForecastHours}">
    <!-- Template dla elementów -->
</CollectionView>
```

4.5. `WindPage`

Opis: Strona prezentująca aktulany kierunek i prędkość wiatru.

Kluczowe Elementy UI:

- **Wyświetlanie prędkości, kierunku i podmuchów wiatru:**

```
<!-- Wind Direction -->

 <VerticalStackLayout Grid.Column="1" VerticalOptions="Center"
HorizontalOptions="Center">
     <Label FontSize="12" Text="Direction" HorizontalOptions="Center" />
     <Label FontSize="16" Text="{Binding Current.WindDir}"
HorizontalOptions="Center" />
 </VerticalStackLayout>

 <!-- Wind Speed -->
 <VerticalStackLayout Grid.Column="3" VerticalOptions="Center"
HorizontalOptions="Center">
     <Label FontSize="12" Text="Speed (m/s)" HorizontalOptions="Center" />
     <Label FontSize="16" Text="{Binding
Current.WindKph,Converter={StaticResource KmHToMsConverter}}"
HorizontalOptions="Center" />
 </VerticalStackLayout>

 <!-- Gusts -->
 <VerticalStackLayout Grid.Column="5" VerticalOptions="Center"
HorizontalOptions="Center">
     <Label FontSize="12" Text="Gust (m/s)" HorizontalOptions="Center" />
     <Label FontSize="16" Text="{Binding
Current.GustKph,Converter={StaticResource KmHToMsConverter}}"
HorizontalOptions="Center" />
 </VerticalStackLayout>
```

---

5. Szczegółowy Opis Funkcjonalności

- **Wyszukiwanie Lokalizacji:** Użytkownik może wprowadzić nazwę miasta w polu wyszukiwania, a aplikacja pobierze i wyświetli dane pogodowe dla tej lokalizacji.
- **Aktualizacja Pogody:** Dane pogodowe są odświeżane automatycznie przy uruchomieniu aplikacji oraz po przeciągnięciu w dół na stronie głównej.
- **Preferencje Użytkownika:** Użytkownik może wybrać preferencję, taką jak "Indoor Activities" czy "Travel", a aplikacja wygeneruje odpowiednią rekomendację.
- **Prognoza Godzinowa i Dzienna:** Aplikacja wyświetla szczegółowe prognozy na najbliższe godziny oraz dni, w tym temperatury, opady, zachmurzenie itp.
- **Zapisywanie Ulubionych Lokalizacji:** Użytkownik może dodawać lokalizacje do listy ulubionych, które są przechowywane w pamięci aplikacji.

---

6. Wnioski i Możliwości Rozwoju

Aplikacja pogodowa oferuje szeroki zakres funkcjonalności, jednak istnieją możliwości jej dalszego rozwoju:

- **Integracja z innymi API pogodowymi** w celu zwiększenia dokładności danych.
- **Dodanie powiadomień push** informujących o nagłych zmianach pogodowych czy alertach.
- **Personalizacja interfejsu użytkownika**, np. poprzez motywy kolorystyczne.
- **Wprowadzenie obsługi wielu języków** dla użytkowników z różnych krajów.
- **Rozszerzenie rekomendacji** o bardziej szczegółowe wskazówki, np. dotyczące ubioru czy aktywności sportowych

7. Licencja ikony