Aleksander Dejewski, 242824, IV rok NSI

Aplikacja mobilna – Pogoda

# Opis aplikacji

Aplikacja będzie służyć do pobierania i wyświetlania pogody użytkownikowi. Prognoza pogody będzie wyświetlana w aplikacji oraz na widżecie na ekranie głównym. Całość będzie napisana w języku C# z wykorzystaniem technologii Xamarin.Forms. Początkowo wspierana będzie platforma Android. Jako, że technologia Xamarin pozwala na współdzielenie sporej ilości kodu pomiędzy platformami to możliwe będzie także uruchomienie aplikacji pod systemem iOS. Będzie to jednak wymagać drobnych korekt w kodzie.

# Opis ekranów aplikacji

## 2.1 Ekran główny aplikacji

Po uruchomieniu aplikacji użytkownikowi zostanie przedstawiony ekran pokazany na zdjęciu nr 1, natomiast ekran nr 2 będzie widoczny po przesunięciu w dół. Ekran główny przedstawia w sposób precyzyjny aktualną lokalizację oraz datę. Dodatkowo najważniejsze informacje dotyczące aktualnego stanu pogody są widoczne w przejrzysty sposób. Na górze ekranu znajduje się pasek wyszukiwania pozwalający na znalezienie kolejnych lokalizacji. Dodatkowo po wybraniu lokalizacji automatycznie jest ona dodawana do zapamiętanych miejsc. Dzięki temu możemy w szybki sposób zobaczyć prognozę pogody dla miejsc, które były wyszukiwane w przeszłości. Dostępna jest także opcja aktualnej pozycji geograficznej z wykorzystaniem GPSu wbudowanego w telefon użytkownika.

Prognoza jest przedstawiana dla aktualnego czasu, ale możemy także podejrzeć prognozę pogody godzinową oraz długoterminową prognozę dla kolejnych dni.

Uwzględniono także takie informacje jak wschód i zachód słońca.

**

*Zdjęcie nr 1. Ekran pogody aplikacji.*



*Zdjęcie nr 2. Ekran pogody aplikacji cd.*

## Ekran powiadomień

Ekran powiadomień jest przedstawiony na zdjęciu nr 3. Służy on do konfiguracji jakiego typu powiadomienia użytkownik chciałby otrzymywać.



*Zdjęcie nr 3. Ekran powiadomień.*

Użytkownik może skonfigurować o której godzinie chciałby otrzymywać powiadomienia. Całość jest konfigurowalna indywidualnie dla każdej lokalizacji. Możemy także skonfigurować powiadomienie dla aktualnej lokalizacji wtedy wymagane jest uwzględnienie to w uprawnieniach dla aplikacji – używanie GPSu podczas, gdy aplikacji jest zamknięta. Przykładowe zdarzenia pogodowe, których wystąpienie może spowodować wysłanie powiadomienia to:

* burza,
* deszcz,
* opady śniegu,
* mróz,
* silny wiatr w tym trąby powietrzne, tornada, itp.,

W tym miejscu trzeba pamiętać, żeby takie powiadomienia były odpowiednio skonstruowane jeśli chodzi o treść. Nie powinniśmy wysyłać więcej niż jednego powiadomienia na lokalizację dziennie. Wysyłanie większej ilości powiadomień może spowodować ignorowanie ich przez użytkownika.

## Ekran ustawień

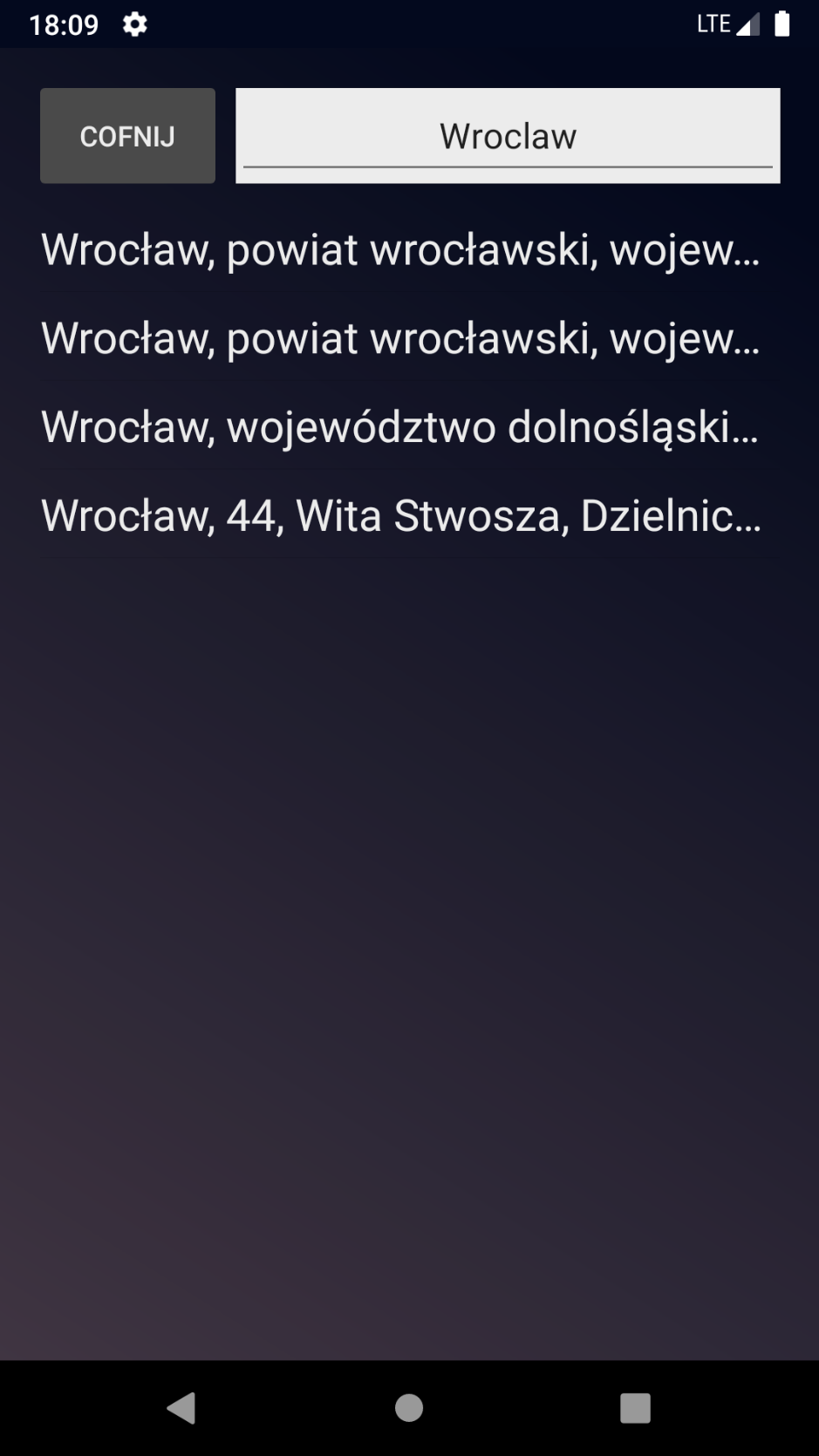
Ekran ustawień pozwala na konfigurację ustawień aplikacji. Na zdjęciu nr 4 pokazano jego wygląd. W tym miejscu mamy możliwość ustawienia podstawowych ustawień takich jak jednostki, w których będzie wyświetlana temperatura, czy prędkość wiatru.



*Zdjęcie nr 4. Ekran ustawień.*

## Ekran wyszukiwania miejsca

Wyszukiwanie miejsca do zapamiętania będzie realizowane na osobnym ekranie, który będzie prezentował znalezione lokalizacje. Po wyborze danej lokalizacji z listy aplikacja będzie przechodziła do głównego ekranu. Na zdjęciu nr 5 pokazano wygląd ekranu wyszukiwania.



*Zdjęcie nr 5. Ekran wyszukiwania.*

## Widget

Dodatkowo w aplikacji będzie dostępny widget pogodowy, będzie on pokazywać na ekranie głównym Androida aktualną pogodę uwzględniając ustawienia użytkownika. Jego wygląd pokazano na zdjęciu nr 6.



*Zdjęcie nr 6. Wygląd widgetu aplikacji.*

Widget może być ustawiony tak, aby pokazywał prognozę pogody dla aktualnej pozycji.

# Opis techniczny

## 3.1 Komunikacja z serwerem

Aplikacja będzie się komunikować z serwerem przy pomocy REST API. Komunikaty będą przesyłane w formacie JSON. Ważne jest, aby w aplikacji uwzględnić przypadek skrajny, gdzie użytkownik nie będzie posiadał dostępu do Internetu. W związku z tym ważne jest wyświetlenie informacji o jego braku. Dodatkowo ważne jest, aby aplikacja zapisywała ostatnio pobrane dane w pamięci wewnętrznej urządzenia, w przypadku braku połączenia z serwerem będzie ona mogła je wyświetlić wraz z informacją, kiedy zostały one pobrane.

Program musi także informować serwer, które powiadomienia chciałby otrzymywać oraz o której godzinie. To także będzie synchronizowane przy użyciu REST API i przesyłane w formacie JSON. Do serwera musi być także przesyłany token, który jest wykorzystywany w komunikacji z API Firebase. Usługa Firebase będzie wykorzystana do przesyłania powiadomień PUSH.

## Powiadomienia

Powiadomienia będą przesyłane do telefonu przy wykorzystaniu Firebase. Backend po stronie serwera o określonej porze wyśle wcześniej skonfigurowane powiadomienia.

## Pamięć wewnętrzna

W aplikacji będzie wykorzystana prosta baza danych SQLite do przechowywania ustawień. Co więcej osobna tabela będzie służyła do przechowywania ostatnio pobranych danych pogodowych. Będą one wykorzystywane w przypadku braku połączenia z serwerem. W bazie danych zapisane będą także ostatnio wyszukiwane miejsca.

## Wyszukiwanie miejsc

Jako, że aplikacja pozwala na wyszukiwanie miejsc trzeba zamienić szukaną nazwę na współrzędne geograficzne. W tym miejscu wykorzystane będzie API do openstreetmap. API to pozwala na taką właśnie zamianę i zwraca listę miejsc co pozwoli użytkownikowi wybrać już odpowiednie miejsce, które on poszukuje.

## Pobieranie pogody

W tym przypadku zostanie wykorzystane API openweathermap. Korzystając z tego API mamy możliwość pobrania pogody dla danego miejsca. Serwis ten pozwala pobrać pogodę na kolejne godziny oraz dni.

# Opis klas

## 4.1. Ekrany

W aplikacji będą 4 ekrany oraz widget dla systemu Android. Zastosowanie technologii jaką jest Xamarin.Forms w projekcie pozwoli na współdzielenie kodu pomiędzy aplikacją Androida oraz iOS. W związku z tym klasy dotyczące ekranów aplikacji będą tworzone we wspólnej części. Widget jako rzecz specyficzna dla danego systemu zostanie jednak napisane w części projektu, która dotyczy tylko systemu Android.

Powstaną 4 klasy opisujące ekrany *NotificationPage*, *SearchPage*, *SettingsPage*, *WeatherPage.*  W przypadku projektów pisanych w Xamarin.Forms zalecane jest użycie tutaj wzorca MVVM (Model-View-ViewModel). Model ten pozwala rozdzielić wygląd aplikacji od jej logiki. W związku, iż zostanie zastosowany właśnie taki wzorzec dla każdej klasy opisującej ekran powstanie także klasa ViewModel odpowiadająca za logikę. Klasy te będę się nazywać odpowiednio: *NotificationViewModel, SearchViewModel, SettingsViewModel, WeatherViewModel*. Jeśli natomiast w widoku występują elementy, które są generowane w listach muszą także powstać klasy Modeli tych elementów. W tym projekcie takie elementy to: pogoda godzinowa, pogoda na kolejne dni, ulubione miejsca, powiadomienia dla konkretnych miejsc. W związku z tym zostaną napisane klasy odpowiadające za te modele odpowiednio: *HourForecase*, *DayForecast*, *FavoritePlace*, *NotificationGroups* oraz *NotificationItem*.

Widget dla systemu Android natomiast wymaga napisania pliku z opisem wyglądu, tutaj *Widget.xml* oraz klasę obsługującą logikę – plik *AppWidgetProvider.cs*.

## Komunikacja z zewnętrznymi serwisami

Za komunikację z serwisem pogodowym OpenWeatherApp będzie odpowiadać klasa *OpenWeather* najważniejsza metoda w tej klasie to *GetWeather* i przyjmujeona parametry długości geograficznej, szerokości geograficznej oraz typu jednostek (metryczne/imperialne). Zwracany będzie obiekt *OpenWeatherOneCall*, który będzie zawierał odebrane dane z serwisu. Zastosowanie będzie tutaj programowanie asynchroniczne przy pomocy słów kluczowych języka C# async/await.

Komunikując się z serwisem OpenStreetMap wykorzystam klasę *Nominatim*, klasa ta będzie posiadać dwie metody. Pierwsza na podstawie wyszukiwanej treści zwróci Listę obiektów typu *Addresses*, które zawierają informacje o wyszukiwanych obiektach. Druga będzie wykorzystywana przy pobieraniu nazwy miejsca z współrzędnych geograficznych i w tym przypadku zwróci pojedynczy obiekt *Addresses*. Tutaj także będzie zastosowane programowanie asynchroniczne przy wykorzystaniu metody async/await.

Ostatnią klasą komunikującą się z zewnętrznym serwisem jest klasa *SyncData*, która odpowiada za synchronizację danych potrzebnych do wysyłania powiadomień z serwisem backendu. Klasa ta będzie posiadać metodę *SyncSettings*, która na podstawie listy ulubionych miejsc wraz z informacją o wysyłanych powiadomieniach oraz czasu wysyłania powiadomienie stworzy pojedynczy obiekt typu JSON, który zostanie wysłany do serwera. Metoda ta będzie zwracać True, bądź False w zależności czy udało się dane zsynchronizować z serwerem, czy też nie.

## Ustawienia aplikacji

Klasa *Settings* jest odpowiedzialna za zapisywanie w preferencjach aplikacji ustawień. Są to m.in. ulubione miejsca (klasa *Place*), czas wysłania powiadomień (klasa *TimeSpan*). Przy synchronizacji ustawień wykorzystywany jest pakiet Xamarin.Essentials, który zawiera klasę *Preferences* pozwalającą na zapis do pamięci wewnętrznej urządzenia niezależnie od systemu operacyjnego.

## Pobieranie lokalizacji urządzenia

Pobieranie lokalizacji urządzenia także będzie realizowane przy użyciu pakietu Xamarin.Essentials i klasy *Geolocation*.

## Wyświetlanie wiadomości wyskakujących – Toast messages

Stworzony zostanie także interfejs IMessage, który będzie implementowany osobno dla systemów iOS i Android przy użyciu metody *DependencyServices*. Pozwala to na wykorzystanie klas specyficznych dla danej platformy. Klasy implementujące ten interfejs to *MessageAndroid* oraz *MesageIOS*.

## Lokalizacja aplikacji – tłumaczenie

Korzystając z plików .resx aplikacja zostanie przetłumaczona na język angielski i polski. Pliki te pozwolą na automatyczne przełączanie się języka aplikacji w zależności od języka ustawionego w systemie operacyjnym.

## Powiadomienia

Korzystając z pakietu nuget Plugin.FirebasePushNotification zostanie zrealizowana rejestracja do serwisu Firebase powiadomień. Obiór powiadomień także będzie zrealizowany korzystając z tej klasy.