Aplikacja Google Maps z technologią rozpoznawania mowy

autor: Bartosz Zysk

Spis treści

1. Wstęp

Żyjemy w czasach w których podróżowanie jest proste oraz wygodne (w chwili pisania owego dokumentu jest to zdanie kontrowersyjne). Ponieważ wszędzie ze sobą zabieramy telefon, stał się on nieodzownym wsparciem w kwesti logistyki oraz turystyki. Posiada on również szereg wielu udogodnień bez których wiele osób nie wyobraża już sobie pracy, czy nawet życia.

Niniejsza aplikacja porusza temat pomocy dla osób chętnych podróży lub po prostu chcących wyszukać miejsca na całym świecie. Drugim zagadnieniem jest ułatwienie owego wyszukiwania.

2. Cel i zakres prac

Celem pracy jest stworzenie aplikacji mobilnej napisanej w języku Java na platformę Android z zaimplementowanym systemem Google Maps tak, aby użytkownik posiadał możliwość wyszukania danego miejsca. Aplikacja będzie również śledziła lokalizację (jeżeli zostanie udzielone odpowiednie pozwolenie) użytkownika.

Celem udogodnienia wyszukiwania, możliwe będzie użycie modułu Rozpoznawania mowy, który dzięki wbudowanemu w urządzenie mikrofonowi będzie mógł „przetłumaczyć” mowę ludzką na odpowiedni ciąg znaków.

Zakres prac obejmuje:

- Przegląd dokumentacji usługi Google Maps
- Przegląd dokumentacji systemu Android w celu zaimplementowania modułu Rozpoznawania mowy
- Zaprojektowanie i stworzenie aplikacji mobilnej
- Testy aplikacji

3. Wykorzystane technologie

Ponieważ pisana aplikacja nie jest programem złożonym, jedynym głównym użytym językiem programowanie jest Java. Do zaimplementowania usługi Google Maps potrzebne jest oczywiście odpowiednie API. Wybranym przeze mnie środowiskiem programistycznym jest Android Studio.

3.1. Java

Java to język programowania zapoczątkowana przez inżynierów z firmy Sun Microsystems w 1991. Język rozwijano i w 1996 wydano pierwszą publiczną implementację. Została zaprojektowana jako język obiektowy, czyli taki w którym możemy definiować pewne abstrakcyjne elementy, które pozwalają lepiej odzwierciedlić elementy naszej aplikacji. Obiekt to element który składa się z pewnych cech, np. obiekt Dom posiada cechę kolor oraz właściciela.[1]

3.2. Android Studio

Android Studio to oficjalny program do tworzenia, testowania oraz modyfikowania oprogramowanie tworzonego na Platformę Android stworzoną przez Google. Środowisko to dominuje rynek na platformę Android a domyślnymi językami w których pisze się aplikacje to Java oraz Kotlin. Posiada on liczne cechy takie jak edytor UI, emulator urządzeń Android oraz liczne rozrzerzenia dostępne do pobrania.[2]

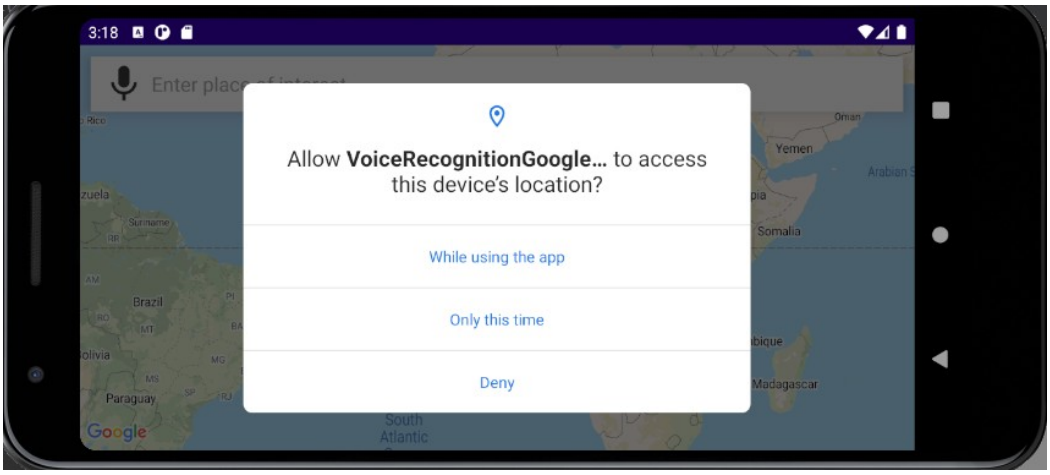
3.3. Google Maps SDK

Google Maps SDK (Software Development kit) to zestaw narzędzi potrzebnych do implementacji wszelakich funkcji dostępnch w systemie Google Maps w aplikacji na platformę Android. Wspomaganymi językami są Java oraz Kotlin. Aby móc używać Google Maps w swoich aplikacjach potrzebny jest klucz API połączony z kontem Google. Przykładowe funkcjonalności usługi to umieszczanie znaczników na mapie, obliczanie najkrótszej trasy do celu, czy chociażby wyświetlanie informacji o danym punkcie zainteresowania. [3]

4. Funkcjonalność aplikacji

Aplikacja przy pierwszym uruchomieniu (bądź kolejnym jeżeli nie uzyskano zgody) pyta się o zezwolenie na pobieranie informacji o lokalizacji urządzenia. Jeżeli użytkownik zgodzi się, na mapie pojawia się mała kropka odpowiadająca lokalizacji. Następnie użytkownik może poruszać się po mapie oraz wyszukać (za pomocą klawiatury, bądź modułu rozpoznawania mowy) miejsca zainteresowania. Po wyszukaniu, mapa zaznacza miejsce (jeżeli udało się go wyszukać) za pomocą znacznika i przybliża widok w jego miejsce.

4.1. Zgoda na pobieranie lokalizacji urządzenia



Rysunek 4.1 Okno pobrania zgody od użytkownika

Gdy użytkownik pierwszy raz uruchomi aplikację (bądź gdy poprzednim razem uruchomił ją i nie dał zgody) pokaże się okno które poprosi użytkownika o udzielenie zgody na pobranie lokalizacji użytkownika. Aplikacja prosi o tzw. *FINE_LOCATION* (istnieje jeszcze *COARSE_LOCATION*) które używa nie tylko WiFi lub mobilnego internetu do sprawdzenia lokalizacji urządzenia, ale również usługi GPS co pozwala na o wiele dokładniejsze określenie lokalizacji.

```
private void getPermission() {
    String[] permission = {Manifest.permission.ACCESS_FINE_LOCATION};
    if (ContextCompat.checkSelfPermission(this.getApplicationContext(), Manifest.permission.ACCESS_FINE_LOCATION)
            == PackageManager.PERMISSION_GRANTED) {
        locationPermission = true;
    } else {
        ActivityCompat.requestPermissions( activity: this, permission, LOCATION_REQUEST_CODE);
    }
    loadMap();
```
Rysunek 4.2 Kod odpowiadający za inicjalizację pobrania zgody

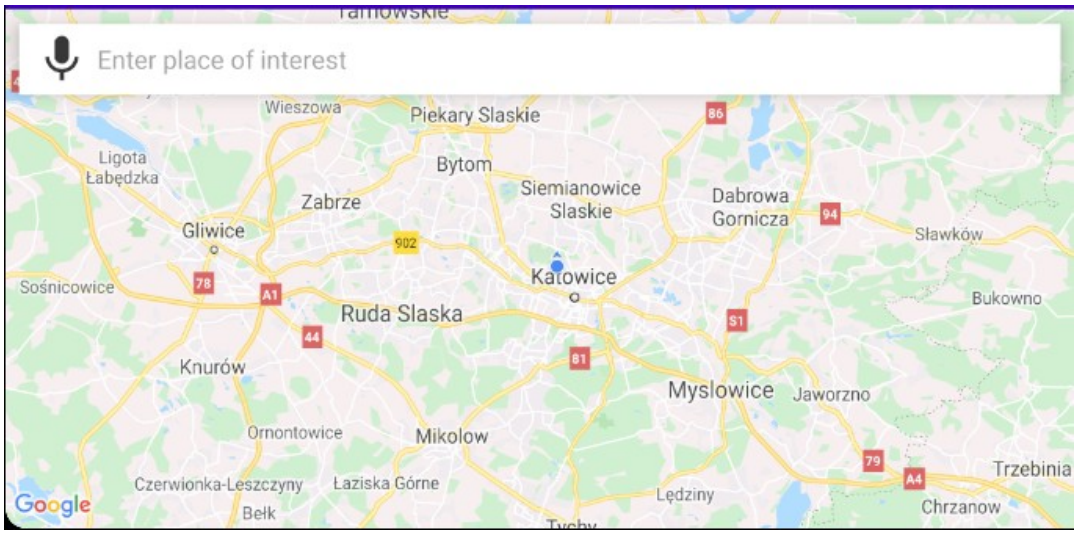
Na rysunku 4.2 kod sprawdza, czy nie została wcześniej udzielona zgoda na pobranie lokalizacji, jeżeli nie, to zostanie wyświetlone okno z opcją jej pobrania.

```
switch (requestCode) {
    case LOCATION_REQUEST_CODE: {
        if (grantResults.length != 0) {
            for (int i = 0; i < grantResults.length; i++) {
                Log.d(TAG, String.valueOf(grantResults[i]));
                if (grantResults[i] != PackageManager.PERMISSION_GRANTED) {
                    locationPermission = false;
                    Toast.makeText( context: this, text: "Permission not granted", Toast.LENGTH_SHORT).show();
                    return;
                }
            }
            locationPermission = true;
        }
    }
}
```

Rysunek 4.3 Kod pobrania zgody

Na rysunku 4.3 widać funkcję która otwiera okno zgody oraz zapisuje w zmiennej *locationPermission* to, czy użytkownik zezwolił na pobranie lokalizacji lub nie.

4.2. Pobranie lokalizacji urządzenia



Rysunek 4.4 Znacznik pokazujący lokalizację urządzenia

Jeżeli uzyskana zostanie zgoda od użytkownika, metoda *getLocation()* zostanie uruchomiona, pobiera ona lokalizacje urządzenia tak, aby aplikacja mogła przybliżyć widok na odpowiedni skrawek mapy za pomocą metodę *moveToPoint()* którą omówię później w tym rozdziale.

```
private void getLocation() {
    userLocation = LocationServices.getFusedLocationProviderClient( activity: this);
    try {
        userLocation.getLastLocation().addOnSuccessListener( activity: this, new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                if (location != null) {
                    moveToPoint(new LatLng(location.getLatitude(), location.getLongitude()), zoom: 12f, name: "Me");
                } else {
                    Toast.makeText( context: MapsActivity.this,
                            text: "Can't load user location, enable localisation feature in your device", Toast.LENGTH_LONG).show();
                }
            }
        });
    } catch (SecurityException e) {
        Log.e(TAG, msg: "Exception: " + e.getMessage());
    }
}
```

Rysunek 4.5 Kod pobierający lokalizację urządzenia

4.3. Wyszukanie punktu zainteresowania

Rysunek 4.6 Pole tekstowe służące do wyszukiwania miejsc

Użytkownik posiada możliwość wyszukania miejsca które go interesuje. Do tego posłużymy się klasą *Geocoder.* Zamienia ona cechę lokalizacji (np. nazwę, adres) na długość oraz szerokość geograficzną. Nie zadziała ona sama, dlatego wspieramy się tutaj usługą Google Places która zawiera bazę danych milionów miejsc na świecie.

```
String search = searchField.getText().toString();

Geocoder geocoder = new Geocoder( context: this);
List<Address> list = new ArrayList<>();
try {
    list = geocoder.getFromLocationName(search, maxResults: 1);
} catch (IOException e) {
    Log.e(TAG, e.getMessage());
}
```

Rysunek 4.7 Kod zwracający adres wyszukanego miejsca

Jak widzimy na rysunku 4.7, klasa *Geocoder* za pomocą metody *getFromLocationName([…])* zapisuje do listy adresów pierwszy wynik na jaki napotka i zamienia go w długość oraz szerokość geograficzną.

```
    if (list.size() > 0) {
        Address address = list.get(0);
        moveToPoint(new LatLng(address.getLatitude(), address.getLongitude()),
                zoom: 12f, address.getAddressLine( index: 0));
    }
    else {
        Toast.makeText( context: this,  text: "Can't find searched place", Toast.LENGTH_LONG).show();
    }
}
```

Rysunek 4.8 Kod pobierający pierwszy znaleziony adres

Jeżeli lista jest pusta, oznacza to, że usługa nie znalazła podanego miejsca i aplikacja ukazje mały komunikat o tym błędzie. Jeżeli znaleziono miejsce, uruchomiona zostaje metoda *moveToPoint([…])* która zostanie omówiona w podrozdziale 4.4.

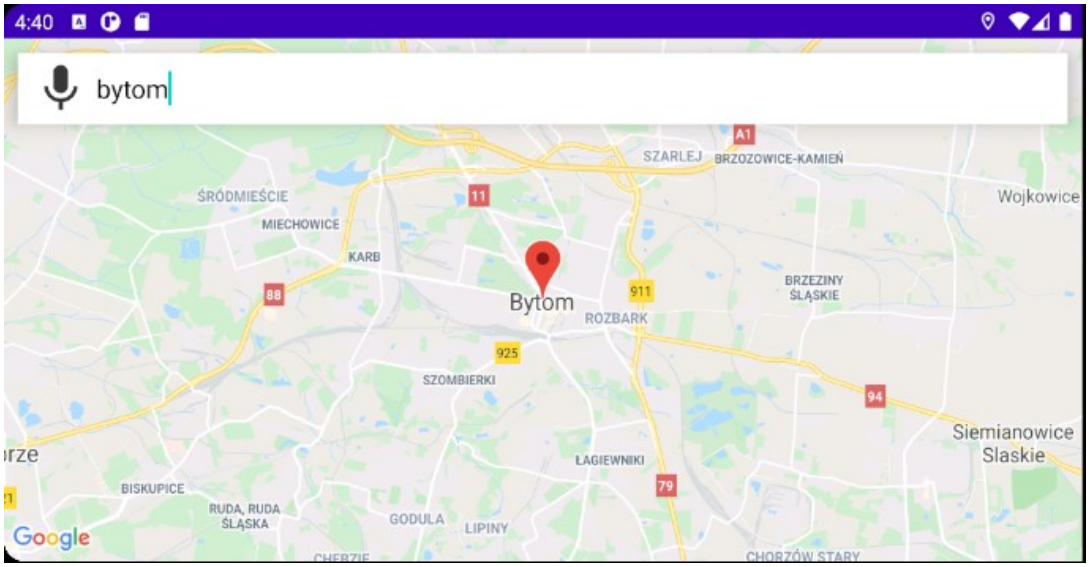
4.4. Znaczniki oraz przybliżanie widoku

```
private void moveToPoint(LatLng latLng, float zoom, String name) {
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, zoom));
    if (markerName!=null) markerName.remove();

    if (name != "Me") {
        MarkerOptions options = new MarkerOptions().position(latLng).title(name);
        markerName = mMap.addMarker(options);
    }
}
```

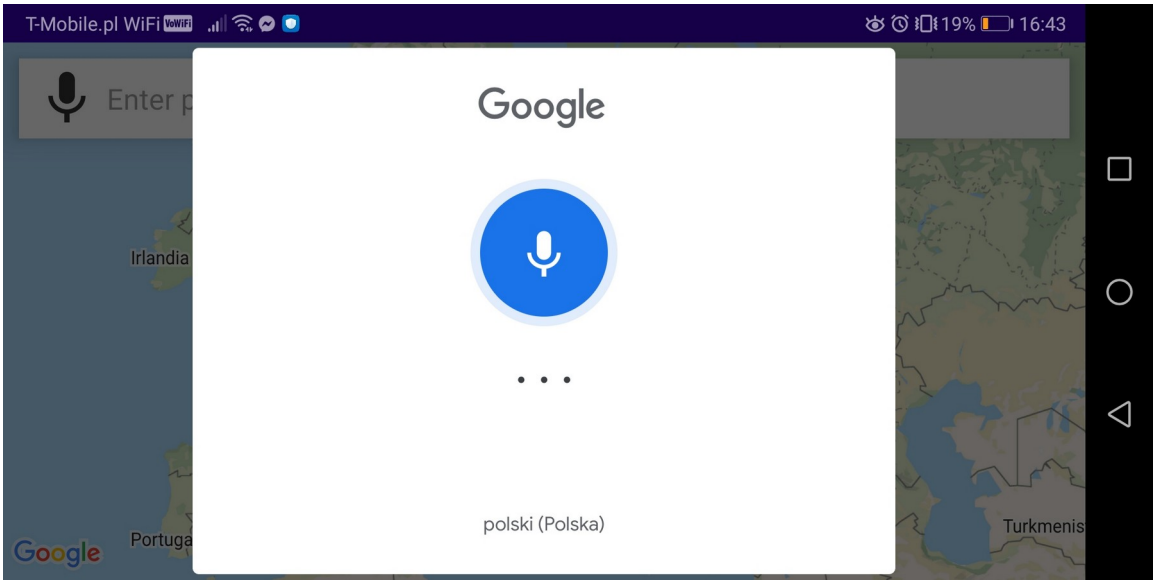
Rysunek 4.9 Kod metody *moveToPoint([…])*

Metoda *moveToPoint([…])* przyjmuje argumenty szerokości i długości geograficznej (*LatLng*), wielkością przybliżenia widoku (*zoom*) oraz parametr *name* który służy do określenia czy aplikacja ma przybliżyć widok na lokalizację urządzenia, czy na wyszukany punkt. Jeżeli jest to punkt, aplikacja dodaje do obiektu mapy google (*mMap*) znacznik w odpowiedmik miejscu za pomocą metody *mMap.addMarker([…])*.



Rysunek 4.10 Znacznik na miejscu wyszukanego punktu

4.5. Moduł rozpoznawania mowy



Rysunek 4.11 Okno zainicjowanego rozpoznawania mowy

Wyszukiwanie za pomocą rozpoznawania mowy inicjujemy za pomocą ikony mikrofonu przy polu wyszukiwania (patrz rysunek 4.6). Aplikacja wyświetla wtedy okno mowy a użytkownik za pomocą wbudowanego w urządzeniu mikrofonu może wyszukać punktu zainteresowania.

```
public void getSpeech(View view) {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());

    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, requestCode: 2);
    } else {
        Toast.makeText( context: this, text: "Speech recognition not supported in this device", Toast.LENGTH
    }
}
```

Rysunek 4.12 Metoda aktywująca moduł rozpoznawania mowy

Rozpoznawanie mowy uruchamiane jest poprzez inicjację obiektu *RecognizerIntent*. *Intent* to „intencja” która odpowiada za obsługę komunikacji między naszą aplikacją a mniejszym komponentem, w tym przypadku modułem obsługi rozpoznawania mowy. Do obiektu *intent* dodajemy również dodatkowe moduły związane z językiem, np. ustawienie języka urządzenia jako języka mowy.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == 2 && data != null) {
        ArrayList<String> result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        searchField.setText(result.get(0));
    }
}
```

Rysunek 4.13 Metoda zamieniająca mowę na ciąg znaków

Następnie uruchamiana jest metoda *onActivityResult([…])* która zamienia wypowiedziane słowa na ciąg znaków i wpisuje je do pola wyszukiwania (patrz rysunek 4.6).

5. Problemy napotkane podczas tworzenia aplikacji

Podczas pisania kodu do aplikacji nie miałem większych trudności, lecz pojawiło się parę problemów oraz pytań na które odpowiedzi i solucje musiałem poszukać w internecie.

5.1. Wiele znaczników po wyszukiwaniu

```
if (markerName!=null) markerName.remove();

if (name != "Me") {
    MarkerOptions options = new MarkerOptions().position(latLng).title(name);
    markerName = mMap.addMarker(options);
}
```

Rysunek 5.1 Kod dodawania oraz usuwania znacznika

Podczas testów aplikacji zauważyłem, iż wyszukanie kolejnego miejsca nie usuwało znacznika umiejscowionego na poprzednim punkcie. Rozwiązaniem tego problemu było utworzenie globalnego obiektu znacznika który posiada metodę *remove()*. Jeżeli obiekt markerName nie jest pusty, na początku metody *moveToPoint([…])* zostaje usunięty, a dodany jest następnie kolejny, odpowiedni co do wyszukanego miejsca.

5.2. Pole wyszukiwania nie uruchamia metod

Po wpisaniu tekstu do pola wyszukiwania, metoda *findPlace()* nie uruchamiała się. Musiałem stworzyć metodę *loadInputField()* która zostaje uruchomiona podczas załadowania się aplikacji (metoda *onCreate([…])*) w której obiekt pola jest zainicjowany i która posiada tzw. *Listener* który przy wyszukaniu uruchamia metodę *findPlace()*.

6. Podsumowanie

Celem pracy było stworzenie aplikacji mobilnej mającej na celu ułatwienie wyszukiwania miejsc na urządzeniu mobilnym przy pomocy usługi Google Maps wraz ze wsparciem modułu rozpoznawania mowy. Projekt napisany został w języku Java w środowisku programistycznym Android Studio. Dzięki prostocie API Google Maps zaimplementowanie podstawowych funkcji mapy nie było wielkim wyzwaniem, za to wynik końcowy jest bardzo przekonujący. Również rozpoznawanie mowy na platformę Android nie sprawia wiele trudności, przez co wprowadzenie go do swojej aplikacji powinno być priorytetem dla wszystkich programistów pracujących na tym systemie.

8. Literatura

[1] https://javastart.pl/baza-wiedzy/wprowadzenie/historia-javy

[2] https://en.wikipedia.org/wiki/Android_Studio

[3] https://developers.google.com/maps/gmp-get-started