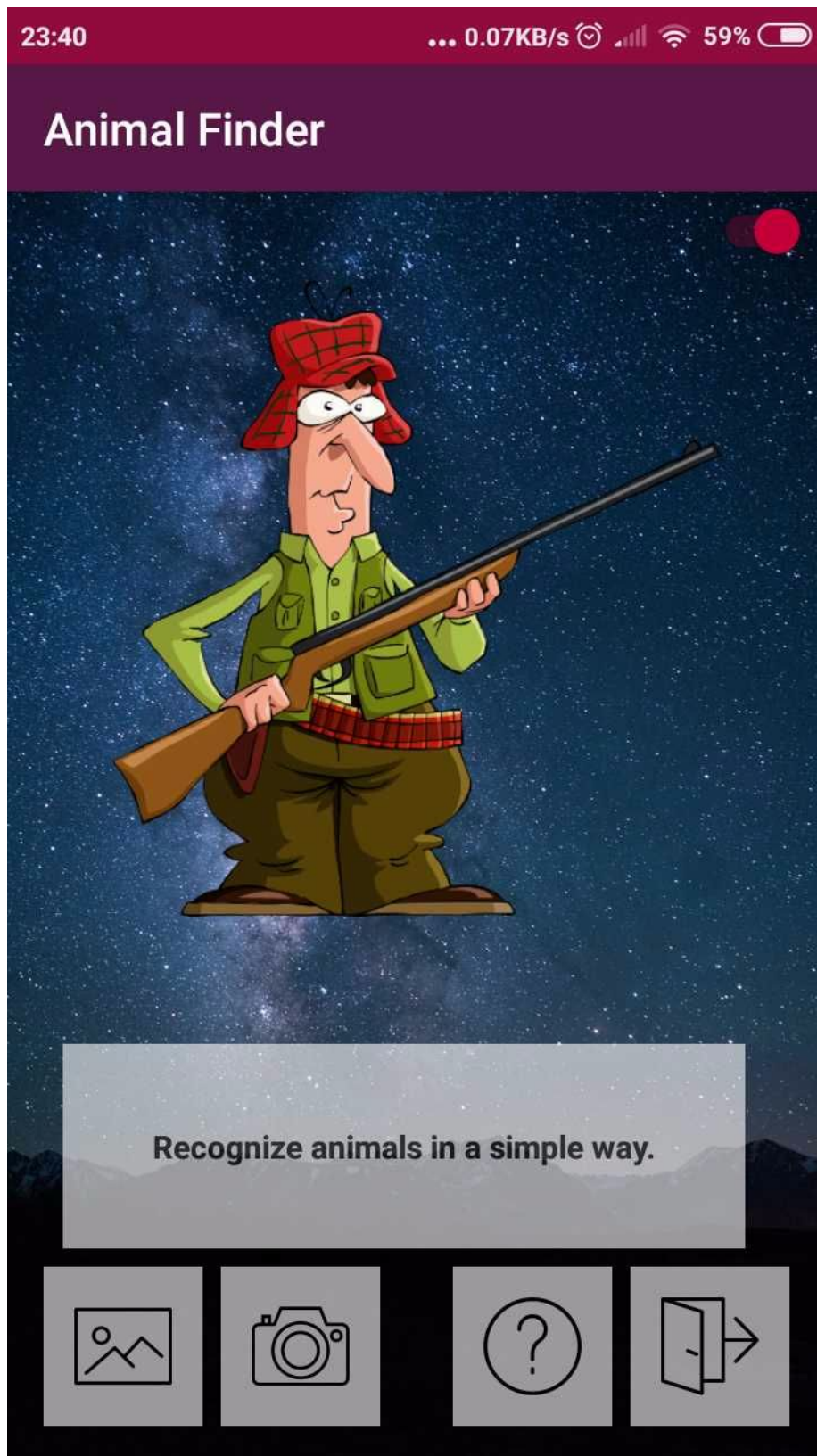


# Animal Finder



# Spis treści

<b>Spis treści</b>	<b>2</b>
<b>Autorzy projektu</b>	<b>3</b>
<b>Cel projektu</b>	<b>3</b>
<b>Wykorzystane Technologie</b>	<b>3</b>
<b>Opis Projektu</b>	<b>3</b>
Przykładowe użycie aplikacji	4
Przypadek 1. Wybranie obrazu z galerii	4
Przypadek 2. Wykonanie zdjęcia	7
<b>Kod aplikacji</b>	<b>10</b>
MainActivity.java	10
AppSingleton.java	12
Wikipedia.java	12

## Autorzy projektu

Paweł Fiołek, Alan Biały, Tomasz Chudzik

## Cel projektu

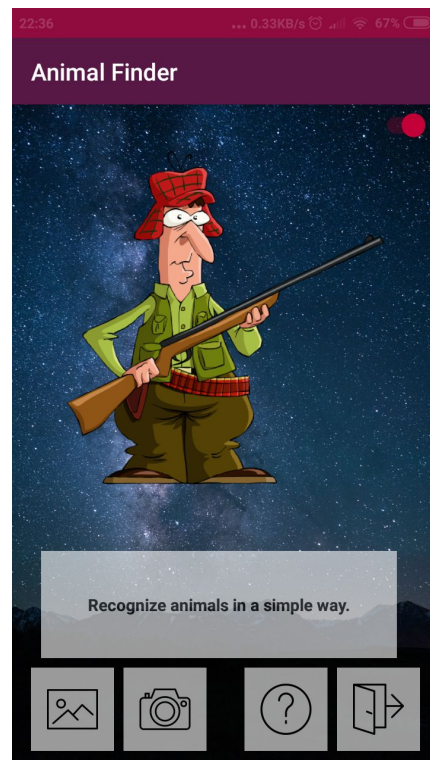
Celem realizowanego projektu było stworzenie aplikacji wykorzystującej mechanizmy rozpoznawania obrazów oraz wyświetlenie opisu rozpoznanego obrazu. Mechanizmy zostały wykorzystane, do rozpoznawania m.in. wizerunków zwierząt.

## Wykorzystane Technologie

Aplikacja została napisana za pomocą oprogramowania Android Studio wykorzystując platformę Firebase do tworzenia aplikacji mobilnych oraz ML Kit dla deweloperów - framework nauczania maszynowego (machine learning).

## Opis Projektu

Po uruchomieniu aplikacji przechodzimy do głównej aktywności, w której zostały umieszczone przyciski nawigacyjne, pozwalające na interakcję z aplikacją oraz pole tekstowe służące do wyświetlania wyników predykcji.

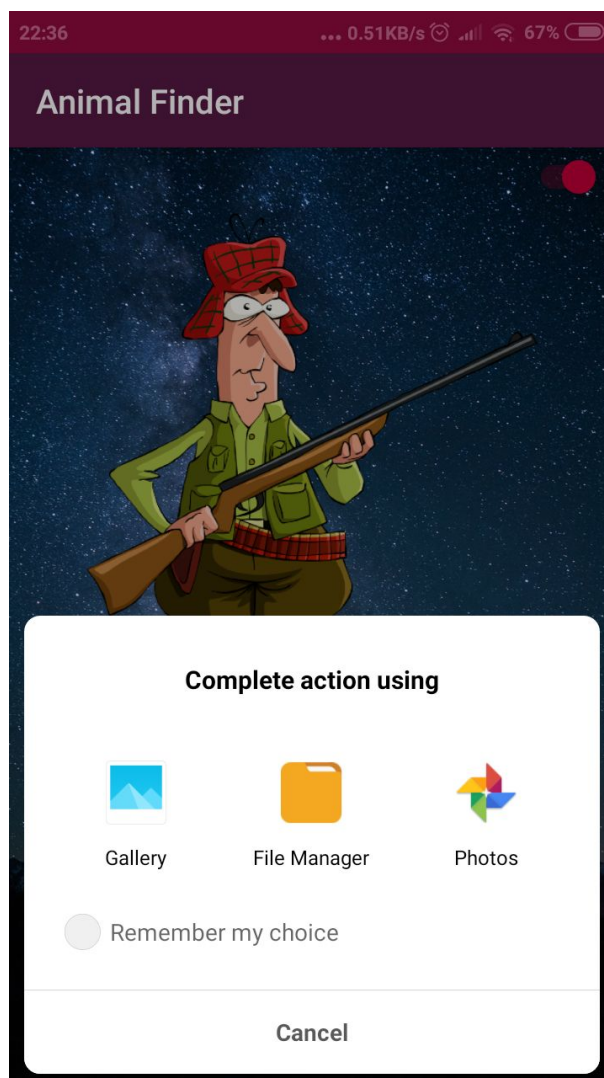


- Przycisk **galerii**
  - pozwala na import obrazu z pamięci wewnętrznej telefonu
- Przycisk **kamery**
  - pozwala na zrobienie zdjęcia korzystając z aparatu w telefonie
- Przycisk **informacji**
  - pozwala na wyświetlenie opisu obrazu
- Przycisk **zamykania aplikacji**
  - pozwala na zamknięcie aplikacji
- Przycisk **zmiany tła** (switch prawy górny róg)
  - pozwala na zmianę tła w aplikacji

## Przykładowe użycie aplikacji

### Przypadek 1. Wybranie obrazu z galerii

- Wybieramy **przycisk galerii**, a następnie dokonujemy wyboru interesującego nas obrazu



- Na ekranie możemy zaobserwować wybrany przez nas **obraz** oraz **wyniki** predykcji dla rozpoznawanego przez nas obrazu. Wyniki posortowane są od najbardziej prawdopodobnych. Każdy wynik składa się z **etykiety** oraz przypisanej do niej **wartości** z przedziału **0.000 - 1.000** oznaczającej prawdopodobieństwo trafności wyniku.






- Wybieramy **przycisk informacji** - aplikacja przenosi nas do nowego widoku w którym znajduję się **opis** szukanego **zwierzęcia** dla najbardziej prawdopodobnego wyniku pobranego ze strony **wikipedia.org**

22:37 ... 3.23KB/s 67%

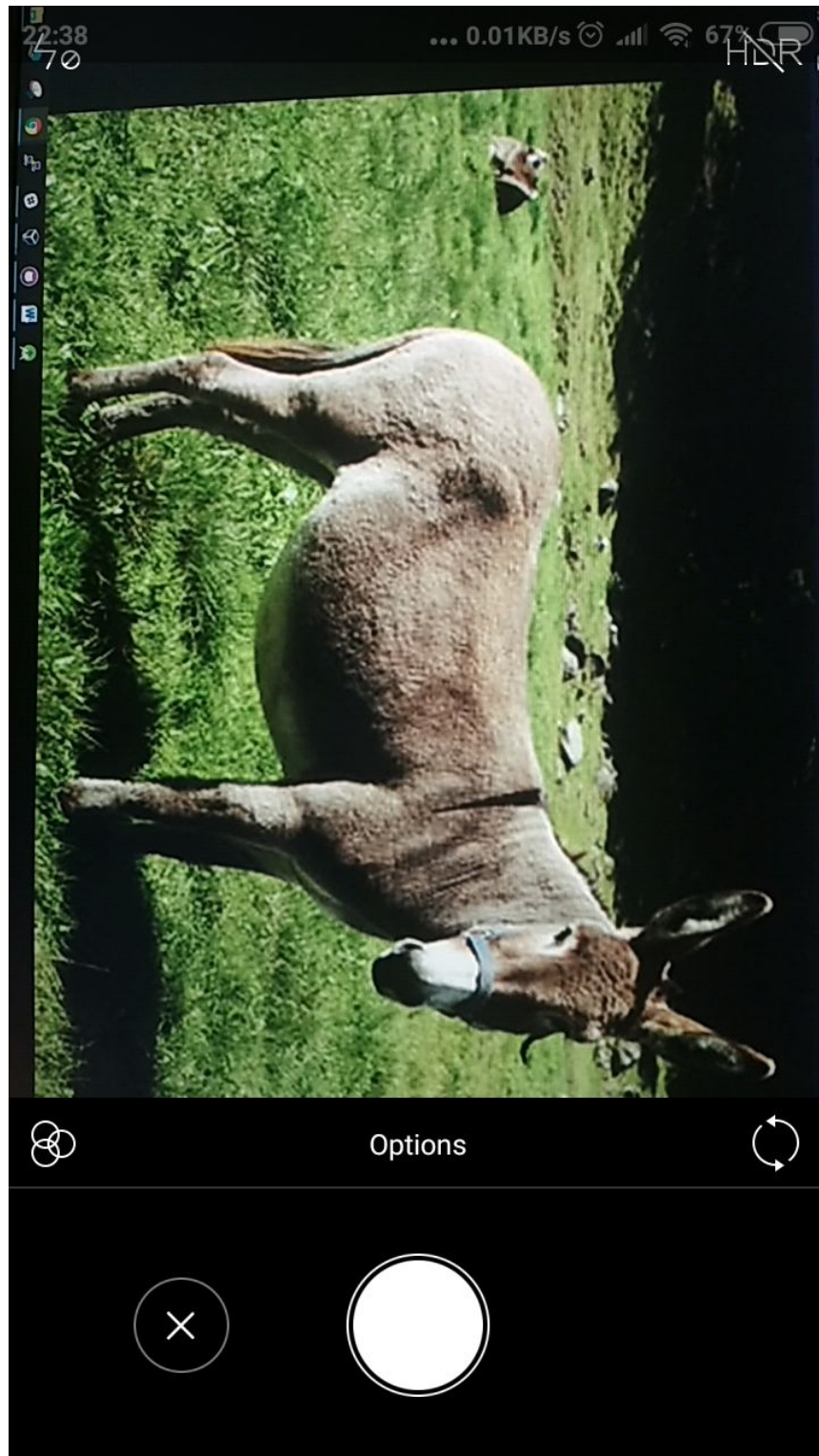
## Animal Finder



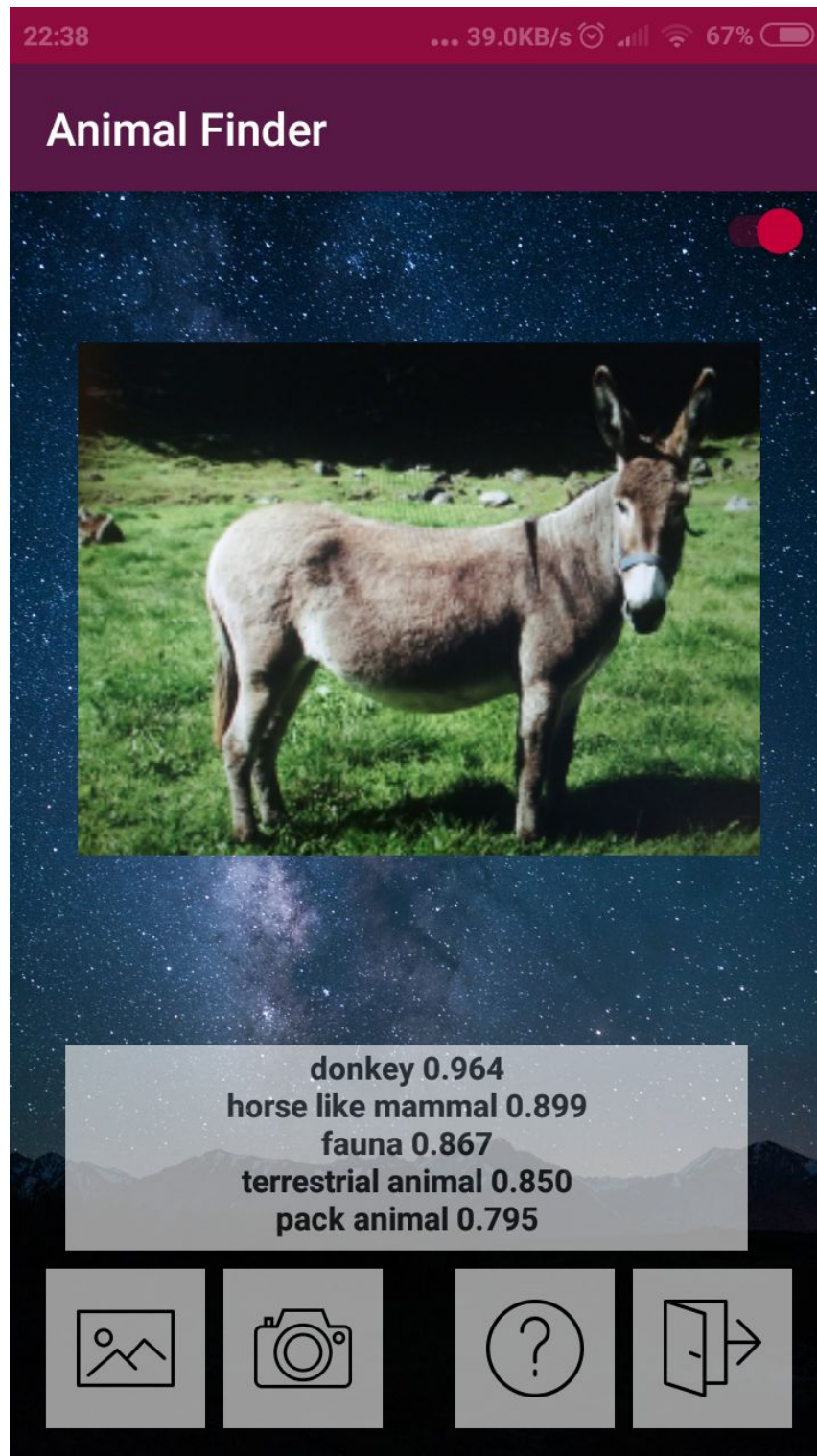
The wolf, also known as the grey/gray wolf or timber wolf, is a canine native to the wilderness and remote areas of Eurasia and North America. It is the largest extant member of its family, with males averaging 43–45 kg (95–99 lb) and females 36–38.5 kg (79–85 lb). It is distinguished from other *Canis* species by its larger size and less pointed features, particularly on the ears and muzzle. Its winter fur is long and bushy and predominantly a mottled gray in color, although nearly pure white, red, and brown to black also occur. Mammal Species of the World, a standard reference work in zoology, recognises 38 subspecies of *C. lupus*.

## Przypadek 2. Wykonanie zdjęcia

- Wybieramy **przycisk zrób zdjęcie** - aplikacja pozwala nam na **wykonanie** zdjęcia wykorzystując nasz **aparat** w telefonie. Na screenie możemy zaobserwować zrobione przez nas **zdjęcie**.



- Na ekranie możemy zaobserwować wybrany przez nas **obraz** oraz **wyniki** predykcji dla rozpoznawanego przez nas obrazu. Wyniki posortowane są od najbardziej prawdopodobnych. Każdy wynik składa się z **etykiety** oraz przypisanej do niej **wartości** z przedziału **0.000 - 1.000** oznaczającej prawdopodobieństwo trafności wyniku.






- Wybieramy **przycisk informacji** - aplikacja przenosi nas do nowego widoku w którym znajduję się **opis** szukanego **zwierzęcia** dla najbardziej prawdopodobnego wyniku pobranego ze strony **wikipedia.org**

22:39

... 0.01KB/s

67%

## Animal Finder



The donkey or ass is a domesticated member of the horse family, Equidae. The wild ancestor of the donkey is the African wild ass, *E. africanus*. The donkey has been used as a working animal for at least 5000 years. There are more than 40 million donkeys in the world, mostly in underdeveloped countries, where they are used principally as draught or pack animals. Working donkeys are often associated with those living at or below subsistence levels. Small numbers of donkeys are kept for breeding or as pets in developed countries.

Powered by wikipedia.org

## Kod aplikacji

Stworzona przez nas aplikacja posiada dwa główne widoki **activity\_main.xml**(domyślny widok po uruchomieniu aplikacji) oraz **activity\_wikipedia.xml**(widok odpowiedzialny za wyświetlanie opisu zwierzęcia). Klasy obsługujące aplikację to **AppSingleton.java**, **Wikipedia.java**, **MainActivity.java**.

- MainActivity.java

*Klasa odpowiada za przechwycenie zdjęcia od użytkownika, zmianę jego rozmiaru oraz jego rozpoznanie.*

- Przechwycenie zdjęcia użytkownika

Realizowane jest w funkcji **onActivityResult()**. Przechwycony obraz konwertowany jest na **Bitmapę**, która później wykorzystywana jest do utworzenia obiektu typu **FirebaseVisionImage**. Obraz przechowywany w takim obiekcie wymagany jest przez detektor znajdujący się chmurze Google pozwalający na rozpoznanie obrazu.

```
121 // ----- onActivityResult -----
122 @Override
123 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
124     super.onActivityResult(requestCode, resultCode, data);
125     if (resultCode == RESULT_OK) {
126         if (requestCode == REQUEST_IMAGE_CAPTURE) {
127             //PHOTO FROM CAMERA
128             bitmapImage = (Bitmap) data.getExtras().get("data");
129             imageDisplay.setImageBitmap(bitmapImage);
130             bitmapImage = resizeImage(bitmapImage);
131             imageFirebase = FirebaseVisionImage.fromBitmap(bitmapImage);
132             textPrediction.setText("Loading...");
133             labelImagesCloud(imageFirebase);
134         } else if (requestCode == REQUEST_IMAGE_GALLERY) {
135             //PHOTO FROM GALLERY
136             Uri uri = data.getData();
137             bitmapImage = null;
138             try {
139                 bitmapImage = MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
140                 imageDisplay.setImageBitmap(bitmapImage);
141             } catch (IOException e) {
142                 e.printStackTrace();
143             }
144             imageDisplay.setImageBitmap(bitmapImage);
145             bitmapImage = resizeImage(bitmapImage);
146             imageFirebase = FirebaseVisionImage.fromBitmap(bitmapImage);
147             textPrediction.setText("Loading...");
148             labelImagesCloud(imageFirebase);
149         }
150     }
151 }
```

- **Zmiana rozmiaru zdjęcia**

Realizowana jest w funkcji **resizeImage()**. Funkcja jest wykorzystywana, aby uniknąć problemu z przepełnieniem buforu podczas przekazywania go do aktywności **wikipedia**. Obraz zmniejszany jest wraz z zachowaniem jego proporcji. Najpierw ustalamy współczynnik proporcji, a następnie skalujemy go przy użyciu metody **createScaledBitmap()**.

```
153 @ private Bitmap resizeImage(Bitmap image) {  
154     float aspectRatio = image.getWidth() /  
155         (float) image.getHeight();  
156     int width = 480;  
157     int height = Math.round(width / aspectRatio);  
158  
159     image = Bitmap.createScaledBitmap(  
160         image, width, height, false);  
161  
162     return image;  
163 }
```

- **Rozpoznanie obrazu**

Realizowane jest w funkcji **labelImagesCloud()**. Funkcja przyjmuje jako parametr obiekt typu **FirebaseVisionImage**, który przechowuje wybrane przez nas zdjęcie. Najpierw tworzymy opcje konfiguracyjne etykiet obrazu wykorzystywane przez nasz detektor tj. wykorzystywany model do rozpoznawania obrazów oraz ilość wygenerowanych wyników predykcji. Następnie tworzymy instancję klasy **FirebaseVisionCloudLabelDetector** zawierającą nasze ustawienia konfiguracyjne. Kolejny krok to utworzenie **Task**, pozwalającego na wykonanie zadania przez naszą aplikację, główna aktywność naszej aplikacji pojawia się w stosie na pierwszym miejscu. Wewnątrz zadania uruchamiany jest nasz **detektor**, który po pomyślnym rozpoznaniu obrazu wypisuje wyniki predykcji wraz z ich nazwami w naszym **textArea(textPrediction)**. W przypadku niepowodzenia(błąd połączenia z API) wyświetlany jest komunikat o błędzie.

```
165 private void labelImagesCloud(FirebaseVisionImage image) {
166     // [START set_detector_options_cloud]
167     FirebaseVisionCloudDetectorOptions options = new FirebaseVisionCloudDetectorOptions.Builder()
168         .setModelType(FirebaseVisionCloudDetectorOptions.LATEST_MODEL)
169         .setMaxResults(5)
170         .build();
171     // [END set_detector_options_cloud]
172
173     // [START get_detector_cloud]
174     FirebaseVisionCloudLabelDetector detector = FirebaseVision.getInstance()
175         .getVisionCloudLabelDetector(options);
176     // [END get_detector_cloud]
177     // [START run_detector_cloud]
178     Task<List<FirebaseVisionCloudLabel>> result =
179         detector.detectInImage(image)
180         .addOnSuccessListener(
181             (OnSuccessListener) (labels) -> {
182                 // Task completed successfully
183                 // [START_EXCLUDE]
184                 // [START get_labels_cloud]
185                 textPrediction.setText("");
186                 for (FirebaseVisionCloudLabel label : labels) {
187                     String text = label.getLabel();
188                     String entityId = label.getEntityId();
189                     float confidence = label.getConfidence();
190                     textPrediction.append(text + " " + String.format("%.3f", confidence) + "\n");
191                     animalName = labels.get(0).getLabel();
192                 }
193                 // [END get_labels_cloud]
194                 // [END_EXCLUDE]
195             })
196         .addOnFailureListener(
197             (e) -> {
198                 textPrediction.setText("An unsuccessful attempt to connect to the server. Check your Internet connection.");
199             });
200     // [END run_detector_cloud]
201 }
202 }
```

- AppSingleton.java

Klasa wykorzystująca bibliotekę **volley** - która odpowiada za wszystko co ma związek z żadaniami sieciowymi w androidzie. Automatycznie planuje zadania takie jak np. pobieranie odpowiedzi z sieci, zapewnia ona przezroczyste buforowanie pamięci. Wykorzystujemy ją do pobrania obiektu json w klasie **Wikipedia.java**

Nazwaliśmy ją Singleton ponieważ pozwala na utworzenie tylko jednej instancji i uzyskaniu dostępu do tej utworzonej.

- Wikipedia.java

Klasa odpowiadająca za pobranie informacji o zwierzęciu przekazanego z **MainActivity**. Informacje pobieramy w formacie json dzięki api dostępnego na wikipedia.org, a następnie wyciągamy opis z obiektu json w postaci tekstu i wyświetlamy go w naszej aktywności.

Realizowane jest to wykorzystując **AppSingleton** - pobieramy jej instancje i dodajemy do kolejki żądań wcześniej utworzony obiekt **JsonObjectReq**, w którym jako parametr podajemy adres **url** i oczekujemy w nim na odpowiedź od api wikipedii. Jest tu realizowana obsługa błędów w przypadku gdy nie będzie informacji o szukanym zwierzęciu lub nie będziemy mieć połączenia z internetem. W pomyślnym przypadku pobrania informacji w formacie json, za pomocą **response.getString("extract")** pobieramy tekst z etykiety extract, w którym znajdują się nasz pożądaný opis. Następnie wyświetlamy go w rozwijanym polu tekstowym.



```
47 // ----- wiki -----
48 public void volleyJsonObjectRequest(String url) {
49
50     String REQUEST_TAG = "volleyJsonObjectRequest";
51
52     JsonObjectRequest jsonObjectReq = new JsonObjectRequest(url, jsonRequest: null,
53     (response) -> {
54         //response.toString();
55         String text = null;
56         try {
57             text = response.getString("name: extract");
58             textViewDescription.setText(text);
59             textViewDescription.append("\n\nPowered by wikipedia.org");
60         } catch (JSONException e) {
61             textViewDescription.setText("Sorry. No information found about " + animalName + " on wikipedia.org");
62         }
63     }, (error) -> {
64         VolleyLog.d("Error: " + error.getMessage());
65         textViewDescription.setText("Sorry. No information found about " + animalName + " on wikipedia.org");
66     });
67
68     // Adding JsonObject request to request queue
69     AppSingleton.getInstance(getApplicationContext()).addToRequestQueue(jsonObjectReq, REQUEST_TAG);
70
71 }
72
73 }
```