ITWS Project

Location Finder App using Android Phone & Cloud Vision API

Parija Malgaonkar : BT18ECE040

Introduction

There are numerous apps on the internet that will help us find the location of a place by using its name. We will be making one such app, but the special feature of this app is that it will let us search for a location with the help of its name (text-input) as well as by tapping on an image of the logo of that place. We will make this possible by using the Cloud Vision API that will help us understand the content of the image. Cloud Vision API enables developers to understand the content of an image by encapsulating powerful machine learning models in an easy way.

Modules of the app:

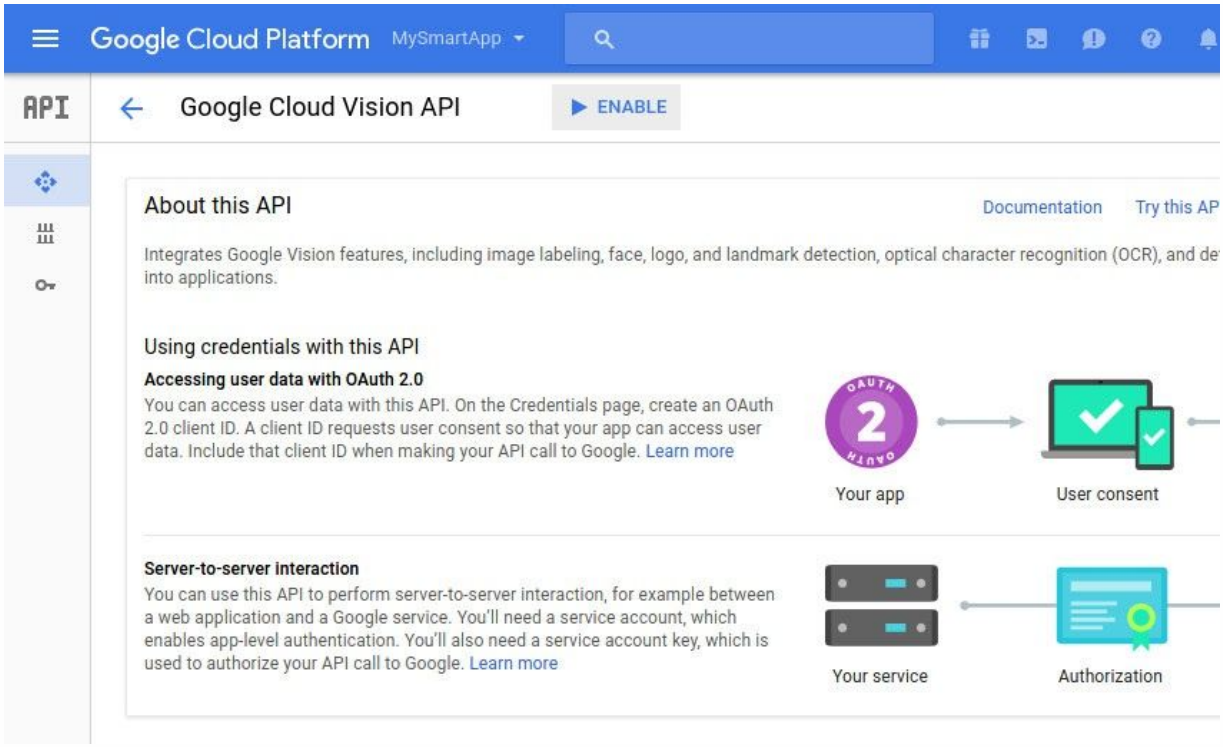
Google Places and Maps : Since our app has the feature of locating places we are searching for, near our location, a good idea is to use the Google Places API. Google Places API is one among the many APIs provided by Google and this is to get geographic information about places using HTTP requests. As for Google Maps, we will be including it in our app solely for its resources. Google Maps has one of the largest databases of all the locations in almost every city of the world, so it is useful to utilise their API and database for our advantage.

Google CloudVision : Google Cloud's Vision API offers the ability to use images such as logos to locate the places the user wants to visit . We can assign labels to images and quickly classify them into millions of predefined categories. The utilisation of CloudVision to make our app more efficient is the main goal of this project.

All these modules in our app will also allow us to save the places that we visit or the places that we like for every individual user, as long as that place exists in these databases. We can also manually enter a place that isn't saved. Now, let us see the working of the app.

CloudVision API :

- To start using the Cloud Vision API in your Android app you have to enable it in the Google Cloud console and acquire a valid API key. So start by logging in to the console and navigating to API Manager > Library > Vision API. In the page that opens, simply press the Enable button.



- Like most other APIs offered by Google, the Cloud Vision API can be accessed using the Google API Client library. To use the library in our app, we add the following `compile` dependencies in the `app` module's **build.gradle** file:

```
compile 'com.google.api-client:google-api-client-android:1.22.0'
compile 'com.google.apis:google-api-services-vision:v1-rev357-1.22.0'
compile 'com.google.code.findbugs:jsr305:2.0.1'
```

- This API enables the ability to look for a location simply by tapping on an image of a logo of that place. This is the main feature of our app. We have to include the CloudVision package in our MainActivity.java file to start :

```
package com.google.sample.cloudvision;
```

- Now, You must configure the Google API client before you use it to interact with the Cloud Vision API. Doing so primarily involves specifying the API key :

```
private static final String CLOUD_VISION_API_KEY = BuildConfig.API_KEY;
```

- Moving further, the HTTP transport, and the JSON factory will be used. As you might expect, the HTTP transport will be responsible for communicating with Google's servers, and the JSON factory will be responsible for converting the JSON-based results the API generates into Java objects.

- For modern Android apps, Google recommends that you use the `NetHttpTransport` class as the HTTP transport and the AndroidJsonFactory class as the JSON factory.The `Vision class` represents the Google API Client for Cloud Vision. Although it is possible to create an instance of the class using its constructor, doing so using the `Vision.Builder` class instead is easier and more flexible.While using the `Vision.Builder` class, you must remember to call the `setVisionRequestInitializer()` method to specify your API key.

```
Vision.Builder visionBuilder = new Vision.Builder(
                new NetHttpTransport(),
                new AndroidJsonFactory(),
                null);

visionBuilder.setVisionRequestInitializer(
            new VisionRequestInitializer("YOUR_API_KEY"));
```

- Once the `Vision.Builder` instance is ready, you can call its `build()` method to generate a new `Vision` instance you can use throughout your app :

```
Vision vision = visionBuilder.build();
```

At this point, we have everything you need to start using the Cloud Vision API.

Working of the app :

- We will start by importing all the google services and client apis from the packages and libraries, that we will need to make sure we can use every function that CloudVision has to offer. We can do this by:

```
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.googleapis.json.GoogleJsonResponseException;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.json.JsonFactory;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.services.vision.v1.Vision;
import com.google.api.services.vision.v1.VisionRequest;
import com.google.api.services.vision.v1.VisionRequestInitializer;
import com.google.api.services.vision.v1.model.AnnotateImageRequest;
import com.google.api.services.vision.v1.model.BatchAnnotateImagesRequest;
import com.google.api.services.vision.v1.model.BatchAnnotateImagesResponse;
import com.google.api.services.vision.v1.model.EntityAnnotation;
import com.google.api.services.vision.v1.model.Feature;
import com.google.api.services.vision.v1.model.Image;
```

- We will ask for permissions from the user's device to access the gallery and use images from there , or to open the camera app and start using it, this will enable us to load the images from which we want to location the place:

```
private static final int GALLERY_PERMISSIONS_REQUEST = 0;
private static final int GALLERY_IMAGE_REQUEST = 1;
public static final int CAMERA_PERMISSIONS_REQUEST = 2;
public static final int CAMERA_IMAGE_REQUEST = 3;\
```

- Using the `VisionRequestInitializer` and the `Vision.builder`, along with the CloudVision API key, we request access to utilise the resources given by the database and use it on the image that we will be processing in the next step.

- By using `BatchAnnotateImagesRequest` multiple image annotation requests are batched into a single service call. This is the Java data model class that specifies how to parse/serialize into the JSON that is transmitted over HTTP when working with the Cloud Vision API.

- Google CloudVision will be called as an Async Task. We go ahead with the `LabelDetectionTask`, which we use for asynchronous batch image annotation and label detection. This means that we use the aforementioned class to annotate the image and detect the label that we are looking for.

- Label detection and Landmark detection are the main features of CloudVision. We use these features after image analysis. We use the `scaleBitmapDown` class to scale down the bitmap to make sure the analysis is not hindered if the image is of a large size.

- We call the CloudVision services using `callCloudVision` for the final bitmap that we will be using for the detection. The response received from the API will be analyzed to provide data in user-readable format. We use `convertResponseToString` to format the response received as a string so that it is readable on the user-end.

Conclusion

We see that we can use the services offered by Google CloudVision to benefit our app and make the usage more efficient. Before, we could find a place by typing in the name, this required us to know the name of the specific location we wanted to visit. Our app enables us to use text, as well as to load an image and select a specific part of the image which includes a symbolization or a logo of the required destination. Then, the app uses Cloud vision API to extract details related to the logo or the entered text by taking help from a database provided by Google and giving us the required result.

References :

- Google CloudVision API Documentation.
- Google Cloud Platform, GitHub page.