package com.vmreddy512.googlemaps;

import android.content.DialogInterface;

import android.content.pm.PackageManager;

import android.location.Location;

import android.location.LocationListener;

import android.os.Bundle;

import android.support.annotation.NonNull;

import android.support.annotation.Nullable;

import android.support.v4.app.ActivityCompat;

import android.support.v4.content.ContextCompat;

import android.support.v7.app.AlertDialog;

import android.support.v7.app.AppCompatActivity;

import android.util.Log;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.Toast;

import com.google.android.gms.common.ConnectionResult;

import com.google.android.gms.common.api.GoogleApiClient;

import com.google.android.gms.location.FusedLocationProviderClient;

import com.google.android.gms.location.LocationServices;

import com.google.android.gms.location.places.GeoDataClient;

//import com.google.android.gms.

import com.google.android.gms.location.places.PlaceDetectionClient;

import com.google.android.gms.location.places.PlaceLikelihood;

import com.google.android.gms.location.places.PlaceLikelihoodBufferResponse;

import com.google.android.gms.location.places.Places;

import com.google.android.gms.maps.CameraUpdateFactory;

import com.google.android.gms.maps.GoogleMap;

import com.google.android.gms.maps.OnMapReadyCallback;

import com.google.android.gms.maps.SupportMapFragment;

import com.google.android.gms.maps.model.CameraPosition;

import com.google.android.gms.maps.model.LatLng;

import com.google.android.gms.maps.model.Marker;

import com.google.android.gms.maps.model.MarkerOptions;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.android.gms.tasks.Task;

public class MapsActivity extends AppCompatActivity

implements GoogleMap.OnMarkerClickListener,

GoogleApiClient.ConnectionCallbacks,

GoogleApiClient.OnConnectionFailedListener, LocationListener,

OnMapReadyCallback {

private static final String TAG = MapsActivity.class.getSimpleName();

private GoogleMap mMap;

private CameraPosition mCameraPosition;

// The entry points to the Places API.

private GeoDataClient mGeoDataClient;

private PlaceDetectionClient mPlaceDetectionClient;

// The entry point to the Fused Location Provider.

private FusedLocationProviderClient mFusedLocationProviderClient;

// A default location (Sydney, Australia) and default zoom to use when location permission is

// not granted.

private final LatLng mDefaultLocation = new LatLng(-33.8523341, 151.2106085);

private static final int DEFAULT\_ZOOM = 15;

private static final int PERMISSIONS\_REQUEST\_ACCESS\_FINE\_LOCATION = 1;

private boolean mLocationPermissionGranted;

// The geographical location where the device is currently located. That is, the last-known

// location retrieved by the Fused Location Provider.

private Location mLastKnownLocation;

// Keys for storing activity state.

private static final String KEY\_CAMERA\_POSITION = "camera\_position";

private static final String KEY\_LOCATION = "location";

// Used for selecting the current place.

private static final int M\_MAX\_ENTRIES = 5;

private String[] mLikelyPlaceNames;

private String[] mLikelyPlaceAddresses;

private String[] mLikelyPlaceAttributions;

private LatLng[] mLikelyPlaceLatLngs;

private static final LatLng PERTH = new LatLng(-31.952854, 115.857342);

private static final LatLng SYDNEY = new LatLng(-33.87365, 151.20689);

private static final LatLng BRISBANE = new LatLng(-27.47093, 153.0235);

private Marker mPerth;

private Marker mSydney;

private Marker mBrisbane;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Retrieve location and camera position from saved instance state.

if (savedInstanceState != null) {

mLastKnownLocation = savedInstanceState.getParcelable(KEY\_LOCATION);

mCameraPosition = savedInstanceState.getParcelable(KEY\_CAMERA\_POSITION);

}

// Retrieve the content view that renders the map.

setContentView(R.layout.activity\_maps);

// Construct a GeoDataClient.

mGeoDataClient = Places.getGeoDataClient(this, null);

// Construct a PlaceDetectionClient.

mPlaceDetectionClient = Places.getPlaceDetectionClient(this, null);

// Construct a FusedLocationProviderClient.

mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClient(this);

// Build the map.

SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()

.findFragmentById(R.id.map);

mapFragment.getMapAsync(this);

}

public void onMapReady(GoogleMap map) {

map.addMarker(new MarkerOptions()

.position(new LatLng(10, 10))

.title("Hello world"));

mMap = map;

// Add some markers to the map, and add a data object to each marker.

mPerth = mMap.addMarker(new MarkerOptions().position(PERTH).title("Perth"));

mPerth.setTag(0);

mSydney = mMap.addMarker(new MarkerOptions()

.position(SYDNEY)

.title("Sydney"));

mSydney.setTag(0);

mBrisbane = mMap.addMarker(new MarkerOptions()

.position(BRISBANE)

.title("Brisbane"));

mBrisbane.setTag(0);

// Set a listener for marker click.

mMap.setOnMarkerClickListener(this);

//

// // Prompt the user for permission.

getLocationPermission();

//

//// // Turn on the My Location layer and the related control on the map.

updateLocationUI();

////

//// // Get the current location of the device and set the position of the map.

getDeviceLocation();

}

/\*\*

\* Called when the user clicks a marker.

\*/

@Override

public boolean onMarkerClick(final Marker marker) {

// Retrieve the data from the marker.

Integer clickCount = (Integer) marker.getTag();

// Check if a click count was set, then display the click count.

if (clickCount != null) {

clickCount = clickCount + 1;

marker.setTag(clickCount);

Toast.makeText(this,

marker.getTitle() +

" has been clicked " + clickCount + " times.",

Toast.LENGTH\_SHORT).show();

}

// Return false to indicate that we have not consumed the event and that we wish

// for the default behavior to occur (which is for the camera to move such that the

// marker is centered and for the marker's info window to open, if it has one).

return false;

}

/\*\*

\* Saves the state of the map when the activity is paused.

\*/

@Override

protected void onSaveInstanceState(Bundle outState) {

if (mMap != null) {

outState.putParcelable(KEY\_CAMERA\_POSITION, mMap.getCameraPosition());

outState.putParcelable(KEY\_LOCATION, mLastKnownLocation);

super.onSaveInstanceState(outState);

}

}

/\*\*

\* Sets up the options menu.

\*

\* @param menu The options menu.

\* @return Boolean.

\*/

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// getMenuInflater().inflate(R.menu.current\_place\_menu, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

// if (item.getItemId() == R.id.option\_get\_place) {

showCurrentPlace();

// }

return true;

}

/\*\*

\* Gets the current location of the device, and positions the map's camera.

\*/

private void getDeviceLocation() {

/\*

\* Get the best and most recent location of the device, which may be null in rare

\* cases when a location is not available.

\*/

try {

if (mLocationPermissionGranted) {

Task<Location> locationResult = mFusedLocationProviderClient.getLastLocation();

locationResult.addOnCompleteListener(this, new OnCompleteListener<Location>() {

@Override

public void onComplete(@NonNull Task<Location> task) {

if (task.isSuccessful()) {

// Set the map's camera position to the current location of the device.

mLastKnownLocation = task.getResult();

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(

new LatLng(mLastKnownLocation.getLatitude(),

mLastKnownLocation.getLongitude()), DEFAULT\_ZOOM));

mMap.addMarker(new MarkerOptions()

.position(new LatLng(mLastKnownLocation.getLatitude(), mLastKnownLocation.getLongitude())));

// .position(new LatLng(.getLatitude(), mLastKnownLocation.getLongitude())));

// mMap = map;

} else {

Log.d(TAG, "Current location is null. Using defaults.");

Log.e(TAG, "Exception: %s", task.getException());

mMap.moveCamera(CameraUpdateFactory

.newLatLngZoom(mDefaultLocation, DEFAULT\_ZOOM));

mMap.getUiSettings().setMyLocationButtonEnabled(false);

}

}

});

}

} catch (SecurityException e) {

Log.e("Exception: %s", e.getMessage());

}

}

/\*\*

\* Prompts the user for permission to use the device location.

\*/

private void getLocationPermission() {

/\*

\* Request location permission, so that we can get the location of the

\* device. The result of the permission request is handled by a callback,

\* onRequestPermissionsResult.

\*/

if (ContextCompat.checkSelfPermission(this.getApplicationContext(),

android.Manifest.permission.ACCESS\_FINE\_LOCATION)

== PackageManager.PERMISSION\_GRANTED) {

mLocationPermissionGranted = true;

// Toast.makeText(this, "Current location:\n" + location, Toast.LENGTH\_LONG).show();

} else {

ActivityCompat.requestPermissions(this,

new String[]{android.Manifest.permission.ACCESS\_FINE\_LOCATION},

PERMISSIONS\_REQUEST\_ACCESS\_FINE\_LOCATION);

}

}

/\*\*

\* Handles the result of the request for location permissions.

\*/

@Override

public void onRequestPermissionsResult(int requestCode,

@NonNull String permissions[],

@NonNull int[] grantResults) {

mLocationPermissionGranted = false;

switch (requestCode) {

case PERMISSIONS\_REQUEST\_ACCESS\_FINE\_LOCATION: {

// If request is cancelled, the result arrays are empty.

if (grantResults.length > 0

&& grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

mLocationPermissionGranted = true;

}

}

}

updateLocationUI();

}

/\*\*

\* Prompts the user to select the current place from a list of likely places, and shows the

\* current place on the map - provided the user has granted location permission.

\*/

private void showCurrentPlace() {

if (mMap == null) {

return;

}

if (mLocationPermissionGranted) {

// Get the likely places - that is, the businesses and other points of interest that

// are the best match for the device's current location.

@SuppressWarnings("MissingPermission") final Task<PlaceLikelihoodBufferResponse> placeResult =

mPlaceDetectionClient.getCurrentPlace(null);

placeResult.addOnCompleteListener

(new OnCompleteListener<PlaceLikelihoodBufferResponse>() {

@Override

public void onComplete(@NonNull Task<PlaceLikelihoodBufferResponse> task) {

if (task.isSuccessful() && task.getResult() != null) {

PlaceLikelihoodBufferResponse likelyPlaces = task.getResult();

// Set the count, handling cases where less than 5 entries are returned.

int count;

if (likelyPlaces.getCount() < M\_MAX\_ENTRIES) {

count = likelyPlaces.getCount();

} else {

count = M\_MAX\_ENTRIES;

}

int i = 0;

mLikelyPlaceNames = new String[count];

mLikelyPlaceAddresses = new String[count];

mLikelyPlaceAttributions = new String[count];

mLikelyPlaceLatLngs = new LatLng[count];

for (PlaceLikelihood placeLikelihood : likelyPlaces) {

// Build a list of likely places to show the user.

mLikelyPlaceNames[i] = (String) placeLikelihood.getPlace().getName();

mLikelyPlaceAddresses[i] = (String) placeLikelihood.getPlace()

.getAddress();

mLikelyPlaceAttributions[i] = (String) placeLikelihood.getPlace()

.getAttributions();

mLikelyPlaceLatLngs[i] = placeLikelihood.getPlace().getLatLng();

i++;

if (i > (count - 1)) {

break;

}

}

// Release the place likelihood buffer, to avoid memory leaks.

likelyPlaces.release();

// Show a dialog offering the user the list of likely places, and add a

// marker at the selected place.

openPlacesDialog();

} else {

Log.e(TAG, "Exception: %s", task.getException());

}

}

});

} else {

// The user has not granted permission.

Log.i(TAG, "The user did not grant location permission.");

// Add a default marker, because the user hasn't selected a place.

mMap.addMarker(new MarkerOptions().title(getString(R.string.cast\_notification\_default\_channel\_name))

.position(mDefaultLocation).snippet(getString(R.string.cast\_notification\_default\_channel\_name)));

// Prompt the user for permission.

getLocationPermission();

}

}

/\*\*

\* Displays a form allowing the user to select a place from a list of likely places.

\*/

private void openPlacesDialog() {

// Ask the user to choose the place where they are now.

DialogInterface.OnClickListener listener = new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

// The "which" argument contains the position of the selected item.

LatLng markerLatLng = mLikelyPlaceLatLngs[which];

String markerSnippet = mLikelyPlaceAddresses[which];

if (mLikelyPlaceAttributions[which] != null) {

markerSnippet = markerSnippet + "\n" + mLikelyPlaceAttributions[which];

}

// Add a marker for the selected place, with an info window

// showing information about that place.

mMap.addMarker(new MarkerOptions()

.title(mLikelyPlaceNames[which])

.position(markerLatLng)

.snippet(markerSnippet));

// Position the map's camera at the location of the marker.

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(markerLatLng,

DEFAULT\_ZOOM));

}

};

}

/\*\*

\* Updates the map's UI settings based on whether the user has granted location permission.

\*/

private void updateLocationUI() {

if (mMap == null) {

return;

}

try {

if (mLocationPermissionGranted) {

mMap.setMyLocationEnabled(true);

mMap.getUiSettings().setMyLocationButtonEnabled(true);

} else {

mMap.setMyLocationEnabled(false);

mMap.getUiSettings().setMyLocationButtonEnabled

(false);

mLastKnownLocation = null;

getLocationPermission();

}

} catch (SecurityException e) {

Log.e("Exception: %s", e.getMessage());

}

}

@Override

public void onConnected(@Nullable Bundle bundle) {

}

@Override

public void onConnectionSuspended(int i) {

}

@Override

public void onLocationChanged(Location location) {

}

@Override

public void onStatusChanged(String s, int i, Bundle bundle) {

}

@Override

public void onProviderEnabled(String s) {

}

@Override

public void onProviderDisabled(String s) {

}

@Override

public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

}

}