```
package com.example.covid_19alertapp.extras;


import android.app.Activity;

import android.content.Context;

import android.content.IntentSender;

import android.location.Location;

import android.os.Looper;

import android.util.Log;


import androidx.annotation.NonNull;


import com.example.covid_19alertapp.roomdatabase.LocalDBContainer;

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

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

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

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

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

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

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

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

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

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

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

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

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

```
import java.util.Calendar;


public abstract class LocationFetch {


    private static Context context;
    private static FusedLocationProviderClient fusedLocationProviderClient;
    private static LocationCallback locationCallback;


    //location request needs to be defined for checkNotificationsSettings() to work
    private static final int MINIMUM_ACCURACY = 20, UPDATE_INTERVAL_MILLIS = 10000;
    private static LocationRequest locationRequest = LocationRequest.create()
            .setInterval(UPDATE_INTERVAL_MILLIS)
            .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);


    public static boolean isLocationEnabled = false;


    // for calculation of distance and time difference
    private static Location prevLocation=null;
    public static void setPrevLocation(Location location){ prevLocation = location; }
    private static int prevMinute = -1;
    // for virtual container
    private static final double CONTAINER_RADIUS = 20.0f; // meters
    private static final int TIME_WINDOW = 15; // minutes
```

```
public static void setup(Context context) {
   /*
   call from context(service) where location fetching is going to start
   setup the variables that require Context
    */

   LocationFetch.context = context;
   fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient(context);

   //location request callback
   locationCallback = new LocationCallback(){

      @Override
      public void onLocationAvailability(LocationAvailability locationAvailability) {
         //called whenever new location is available
         super.onLocationAvailability(locationAvailability);
         Log.d(LogTags.Location_TAG, "onLocationAvailability: "+locationAvailability.toString());
      }

      @Override
      public void onLocationResult(LocationResult locationResult) {
         super.onLocationResult(locationResult);
```

```
            if(locationResult == null){
                //can be null(according to doc)
                Log.d(LogTags.Location_TAG, "onLocationResult: null location");
                return;
            }

            for (Location location : locationResult.getLocations()) {
                // location received do stuff

                if(location.getAccuracy() <= MINIMUM_ACCURACY){
                    // desired location received

                    if(significantDifference(location) || timeWindowExceeded(Calendar.getInstance())) {
                        // location is more than 50m away from 'prevLocation'
                        // or, time difference is more than 15 minutes

                        Log.d(LogTags.Location_TAG,
                                "onLocationResult: time window check =
"+timeWindowExceeded(Calendar.getInstance())
                                        +" distance check= "+significantDifference(location)
                        );

                        // get current time
                        Calendar cal = Calendar.getInstance();
                        //TODO: add year
```

```
                String dateTime = (cal.get(Calendar.MONTH)+1) +"-" // Calender.MONTH is 0 based -_-
why tf?
                        + cal.get(Calendar.DATE) +"-"
                        + cal.get(Calendar.HOUR_OF_DAY);

                Log.d(LogTags.Location_TAG,
                        "addToLocalDB: location received at "
                                + dateTime
                                +" \n["+location.getLatitude()+", "+location.getLongitude()+"]"
                );

                prevLocation = location;
                prevMinute = cal.get(Calendar.MINUTE);

                // add to local storage
                LocalDBContainer.addToLocalDB(location, dateTime, LocationFetch.context);

            }

        }
    }
};

// just to be safe
```

```
    prevLocation = null;
    prevMinute = -1;

}

private static boolean significantDifference(Location location) {
    /*
    returns true if distance between 'prevLocation' and 'location' is more than 20m
     */

    double distance, lat1, long1, lat2, long2;

    if(prevLocation==null) {
        // first time
        return true;
    }

    lat1 = Math.toRadians(prevLocation.getLatitude());
    long1 = Math.toRadians(prevLocation.getLongitude());
    lat2 = Math.toRadians(location.getLatitude());
    long2 = Math.toRadians(location.getLongitude());

    distance = Math.acos(
            Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2)*Math.cos(long1-long2)
    ) * 6371000.0f;
```

```
    distance = Math.abs(distance);

    Log.d(LogTags.Location_TAG, "significantDifference: distance = "+distance);

    if(distance>=CONTAINER_RADIUS) {
        return true;
    }

    return false;
}

private static boolean timeWindowExceeded(Calendar currTimeInstance){
    /*
    return true if more than 15 minutes passed
     */

    int currMinute =  currTimeInstance.get(Calendar.MINUTE);
    Log.d(LogTags.Location_TAG, "timeWindowExceeded: curr minute = "+currMinute);

    if(prevMinute==-1)
        // first time
        return true;

    if(currMinute<prevMinute)
```

```
            currMinute+=60;


        if(currMinute-prevMinute>=TIME_WINDOW)
            return true;


        return false;


    }


    public static void checkDeviceLocationSettings(final Activity activity) {
        /*
        check and prompt the user to enable required location settings
         */


        LocationSettingsRequest.Builder builder = new
LocationSettingsRequest.Builder().addLocationRequest(locationRequest);
        builder.setAlwaysShow(true);


        SettingsClient client = LocationServices.getSettingsClient(activity);
        Task<LocationSettingsResponse> task = client.checkLocationSettings(builder.build());


        task.addOnSuccessListener(activity, new OnSuccessListener<LocationSettingsResponse>() {
            @Override
            public void onSuccess(LocationSettingsResponse locationSettingsResponse) {
                LocationFetch.isLocationEnabled = true;
                Log.d(LogTags.Location_TAG, "checkSettings onSuccess: location update requested");
```

```
    }
});

task.addOnFailureListener(activity, new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {

        Log.d(LogTags.Location_TAG, "checkSettings onFailure: location settings failed");

        LocationFetch.isLocationEnabled = false;

        if(e instanceof ResolvableApiException){
            try{

                ResolvableApiException resolvable = (ResolvableApiException) e;

                resolvable.startResolutionForResult(activity,
                        Constants.LOCATION_CHECK_CODE); //runs onActivityResult() callback of the
associated Activity

            }catch (IntentSender.SendIntentException sendEx){
                //ignore
                Log.d(LogTags.Location_TAG, "onFailure: ignore?");
            }
        }
    }
```

```
        });

    }

    public static void startLocationUpdates() {
        //start the location update
        fusedLocationProviderClient.requestLocationUpdates(
                locationRequest,
                locationCallback,
                Looper.getMainLooper()
        );

        Log.d(LogTags.Location_TAG, "startLocationUpdates: location update started");
    }

    public static void stopLocationUpdates(){
        //remove location update
        fusedLocationProviderClient.removeLocationUpdates(locationCallback);

        Log.d(LogTags.Location_TAG, "stopLocationUpdates: location update stopped");
    }


}
```