```
package com.example.covid_19alertapp.services;


import android.content.Context;
import android.content.Intent;
import android.util.Log;


import androidx.annotation.NonNull;
import androidx.work.Worker;
import androidx.work.WorkerParameters;


import com.example.covid_19alertapp.activities.ShowMatchedLocationsActivity;
import com.example.covid_19alertapp.activities.TrackerSettingsActivity;
import com.example.covid_19alertapp.extras.Constants;
import com.example.covid_19alertapp.extras.LogTags;
import com.example.covid_19alertapp.extras.Notifications;
import com.example.covid_19alertapp.roomdatabase.LocalDBContainer;
import com.example.covid_19alertapp.roomdatabase.VisitedLocations;
import com.example.covid_19alertapp.roomdatabase.VisitedLocationsDao;
import com.example.covid_19alertapp.roomdatabase.VisitedLocationsDatabase;
import com.example.covid_19alertapp.sharedPreferences.MiscSharedPreferences;
import com.example.covid_19alertapp.sharedPreferences.SettingsSharedPreferences;
import com.example.covid_19alertapp.sharedPreferences.UserInfoSharedPreferences;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseException;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.Calendar;
import java.util.List;

import static android.content.Context.MODE_PRIVATE;

/*
   Performs background tasks:
      (1) if location not allowed
            notify and ask to allow,
      (2) delete 7 days old locations in local database,
      (3) query firebase with local data to find match and notify immediately if match found
*/

public class BackgroundWorker extends Worker {

   // stop loop, Bangla niyome listener shorao TODO: kaj korena
   private boolean matchFound;

   // firebase reference and listener
   private DatabaseReference refToMatch, refToMatchHome;
   private ValueEventListener findMatch = new ValueEventListener() {
      @Override
      public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

         if(dataSnapshot.getValue()!=null){
            // INFECTED LOCATION MATCH FOUND!

            // remove turn location on prompt
```

```
                Notifications.removeNotification(Constants.PromptTrackerNotification_ID,
getApplicationContext());

                // open ShowMatchedLocationsActivity on notification tap
                Intent notificationIntent = new Intent(getApplicationContext(),
ShowMatchedLocationsActivity.class);

                // show notification
                Notifications.showNotification(
                        Constants.DangerNotification_ID,
                        getApplicationContext(),
                        notificationIntent,
                        true
                );

                Log.d(LogTags.Worker_TAG, "onDataChange: match found. notified.");

                // try to break loop
                matchFound = true;

                // remove listener after finding any match (will show all through another activity)
                refToMatch.removeEventListener(findMatch);

            }

        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
```

```
        Log.d(LogTags.Worker_TAG, "onCancelled: no internet? "+databaseError.getMessage());
    }
};

public BackgroundWorker(@NonNull Context context, @NonNull WorkerParameters workerParams) {
    super(context, workerParams);
}

@NonNull
@Override
public Result doWork() {

    if(!SettingsSharedPreferences.getLocationTrackerState(getApplicationContext()) && isDayTime() ) {
        // tracker is off prompt notification

        Intent notificationIntent = new Intent(getApplicationContext(), TrackerSettingsActivity.class);

        Notifications.createNotificationChannel(getApplicationContext());
        Notifications.showNotification(
                Constants.PromptTrackerNotification_ID,
                getApplicationContext(),
                notificationIntent,
                true
        );
    }


    // query home
```

```
        queryHomeLocation();


        //TODO:[CHECK] delete 7 days old locations from room db


        // local db
        VisitedLocationsDatabase roomDatabase =
    VisitedLocationsDatabase.getDatabase(getApplicationContext());
        VisitedLocationsDao visitedLocationsDao = roomDatabase.visitedLocationsDao();


        // delete seven days ago entries
        visitedLocationsDao.deleteSevenDaysAgoVisitedLocations
                ("%"+dateLastWeek()+"%");



        /// QUERY FIREBASE


        // initialize as not found
        matchFound = false;


        // firebase configs
        try{
            // can do this only at first time invocation of 'FirebaseDatabase.getInstance()'
            // lem -_-
            FirebaseDatabase.getInstance().setPersistenceEnabled(true);
        }catch (DatabaseException e){
            Log.d(LogTags.Worker_TAG, "doWork: firebase setPersistent issue. ki korbo ami ekhon?");
        }


        refToMatch = FirebaseDatabase.getInstance().getReference();
        // fetch from local db and query firebase

        List<VisitedLocations> localLocationsList = visitedLocationsDao.fetchAll();


        Log.d(LogTags.Worker_TAG, "doWork: local db fethced");


        for (VisitedLocations currentEntry: localLocationsList)

        {


            if(matchFound)

                break;


            // format = "latLon_dateTime"

            String[] splitter = currentEntry.splitPrimaryKey();


            // firebase query values

            String key = currentEntry.getATencodedlatlon();

            String dateTime = splitter[1];


            Log.d(LogTags.Worker_TAG, "doWork: Query-> key = "+key+" dateTime = "+dateTime);


            // query in firebase

            refToMatch =
FirebaseDatabase.getInstance().getReference().child("infectedLocations").child(key).child(dateTime);

            refToMatch.addListenerForSingleValueEvent(findMatch);


            try {

                Thread.sleep(1000);

            } catch (InterruptedException e) {
```

```
            Log.d(LogTags.Worker_TAG, "doWork: "+e.getMessage());
        }

    }



    Log.d(LogTags.Worker_TAG, "doWork: worker WORKED!");

    return Result.success();
}

private boolean isDayTime() {

    int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);

    // 7AM to 11PM
    return (hour>=7 && hour<=23);
}

private void queryHomeLocation() {

    // firebase configs
    try{
        // can do this only at first time invocation of 'FirebaseDatabase.getInstance()'
        // lem -_-
        FirebaseDatabase.getInstance().setPersistenceEnabled(true);
    }catch (DatabaseException e){
        Log.d(LogTags.Worker_TAG, "doWork: firebase setPersistent issue. ki korbo ami ekhon?");
    }
```

```
        List<String> queryKeys;


        String homeLatLng = UserInfoSharedPreferences.getHomeLatLng(getApplicationContext());

        if(homeLatLng.equals("")){

            Log.d(LogTags.Worker_TAG, "queryHomeAddress: why the hell is home null");

            return;

        }


        String[] latLng = homeLatLng.split(",");


        queryKeys = LocalDBContainer.calculateContainer(Double.parseDouble(latLng[0]),
Double.parseDouble(latLng[1]), "Bangladesh");


        for (String query: queryKeys) {


            if(matchFound)

                break;


            // need '@' instead of '.'

            query = query.replaceAll("\\.","@");


            Log.d(LogTags.Worker_TAG, "queryHomeLocation: home query = "+query);


            refToMatchHome =
FirebaseDatabase.getInstance().getReference().child("infectedHomes").child(query);

            refToMatchHome.addListenerForSingleValueEvent(findMatch);


        }
```

```
}


private String dateLastWeek(){

    String date = "";

    int currMonth = Calendar.getInstance().get(Calendar.MONTH)+1;
    int currDate = Calendar.getInstance().get(Calendar.DATE);

    int resDate = currDate - 7, resMonth = currMonth;

    if(resDate<=0){

        switch (currMonth){

            case 1:

                resMonth = 12;
                resDate = 31+resDate;

                break;

            case 2:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                resDate = 30+resDate;
                resMonth = currMonth-1;

                break;

            case 3:

                //TODO: add leap-year check
                resDate = 28+resDate;
                resMonth = currMonth-1;

                break;

            default:
                resDate = 31+resDate;
                resMonth = currMonth-1;
        }

    }

    date = resMonth+"-"+resDate;

    Log.d(LogTags.Worker_TAG, "dateLastWeek: seven days ago = "+date);

    return date;
}
```

```
    @Override
    public void onStopped() {
        super.onStopped();

        Log.d(LogTags.Worker_TAG, "onStopped: Worker stopped. why?");

        // set shared preference false
        MiscSharedPreferences.setBgWorkerStatus(getApplicationContext(), false);
    }
}
```