ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645
Indian Journal of Science and Technology, *Vol 11(20), DOI: 10.17485/ijst/2018/v11i20/123346, May 2018*

Cloud Based Android App Data Transmission (Leverage Connectivity)

Sumendra Yogarayan*, Afizan Azman, Tan Geok Huei, Kirbana Jai Raman, Siti Fatimah Abdul Razak, Mohd Fikri Azli Abdullah, Siti Zainab Ibrahim, Anang Hudaya Muhamad Amin and Kalaiarasi Sonai Muthu

Faculty of Information Science and Technology, Multimedia University, Malacca, Malaysia; mastersumen@gmail.com, afizan.azman@mmu.edu.my, yihui_tan92@hotmail.com, jpk_kirbz@hotmail.com, fatimah.razak@mmu.edu.my, mfikriazli.abdullah@mmu.edu.my, fatimah.razak@mmu.edu.my, anang.amin@mmu.edu.my, kalaiarasi@mmu.edu.my

Abstract

Objective: This paper intents to develop an Android Based Application integrating with backend cloud services instead of local storage database. **Methods**: In this paper, wepropose a mechanism in the context of a mobile application with cloud services integration to identify distance. The aim of the proposed work is to identify distance between each devices and triggers an alert if there are any nearby devices approaching. The mechanism used for calculating the distance is based on distance Between. **Findings**: Mobile application that are based on geo-fence has been in the market in recent years. Most of the current geo-fence application are providing direction and estimation of distance to reach their desired location. However, the constraint is that, there is no framework or mechanism that is dedicated for identifying the distance between each devices and triggers an alert. As of late, mobile application has been the leverage connectivity for vehicles. **Application/Improvements:** The future enhancement for the application to be based on vehicle distance monitoring and traffic information.

Keywords: Alert, Android App, Distance Estimation, Leverage Connectivity

1. Introduction

The advancement of mobile application has involved many aspects to vehicle routine. Nowadays, users in the vehicle are able to access the Internet while they are on the road. Gadgets are fitted in vehicles that can be easily access the travelling information, such as maps, traffic record and many more. Commonly in vehicle, connectivity can be divided into two, which are the embedded connectivity where gadgets are built in the vehicles whereas leverage connectivity is any gadgets that are brought into the vehicles. There are many existing mobile apps that measure distance based on user location (from the current place to the destination), for example Google Map, it only offers users to check on their current location and plan the route from a place to another place based on the transportation method they prefer to ride. However, there is no existing application that can identify the nearby vehicles or devices and their distances between one another. Even though in Google Map, there is no such feature available in it.

Besides, the cloud services not really integrated with such mobile apps. Most of the current mobile apps generally are using the embedded database instead of using the cloud database. Embedded database such as SQLite is used due to its lightweight, self-contained librar-

*Author for correspondence

ies with no server component and small code footprint features and all the databases are locally on the device. Furthermore, android app and cloud services are barely visible to end users of how data is transmitted. Generally, the data transmission and processing process are carrying out in the backend. Hence, it is impossible for user to directly monitor all the processes involving the data transmission from the app itself to the backend database especially through the cloud services.

The scope of this project is to design and develop an android-based tracking app for measuring the distance between the vehicles or user devices and to determine and understand how the location values (data) can be transmitted between android-based tracking app and the backend cloud database. The tracking app will triggers an alert to the users if any nearby vehicles or devices approaching in the specific range of distance. The next scope is to study the cloud infrastructure environment and make use of this knowledge to identify the suitable cloud services that can facilitate the transmission of the (location values) data through the tracking app.

2. Findings

The distance calculation method is a vital part in the whole HiTracker app. There are several distance calculation methods available in the online resources, which will be taken for the comparison in order to identify which method is suitable to use in the HiTracker app. The three available distance calculation methods are distanceTo method, distanceBetween method, and the Haversine formula.

2.1 distanceTo Method

distanceTo is one of the methods which is under Location class defined in Android Platform. distanceTo method returns the approximate distance in meters between this (current) the location and the given location. Distance is defined using the WGS84 ellipsoid, which is the reference system for the Global Positioning System (GPS). Figure 1, illustrates the sample code snippet of the distanceTo method. The float data type will return the approximate distance in meters[1].

```
Location loc1 = new Location("");
loc1.setLatitude(lat1);
loc1.setLongitude(lon1);

Location loc2 = new Location("");
loc2.setLatitude(lat2);
loc2.setLongitude(lon2);

float distanceInMeters = loc1.distanceTo(loc2);
```

Figure 1. Sample Code Snippet for distanceTo method.

2.2 distanceBetween Method

distanceBetween is another method which is also under the category of Location class in Android Platform. distanceBetween computes the approximate distance in meters between two locations, and optionally the initial and final bearings of the shortest path between them.

distanceBetween

```
void distanceBetween (double startLatitude,
                      double startLongitude,
                      double endLatitude,
                      double endLongitude,
                      float[] results)
```

Figure 2. distanceBetween syntax.

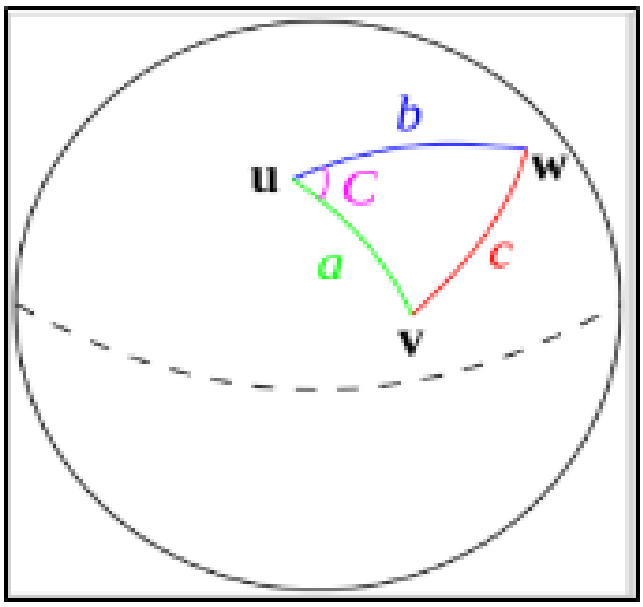
Parameters	
`startLatitude`	`double`: the starting latitude
`startLongitude`	`double`: the starting longitude
`endLatitude`	`double`: the ending latitude
`endLongitude`	`double`: the ending longitude
`results`	`float`: an array of floats to hold the results

Figure 3. Parameters listings for distanceBetween syntax.

Distance is defined using the WGS84 ellipsoid. In Figure 2, illustrates the syntax of the distanceBetween method. The computed distance is stored in results. If results has length 2 or greater, the initial bearing is stored in results. If results has length 3 or greater, the final bearing is stored in results. In Figure 3, lists the parameters for the distanceBetween syntax.

2.3 Haversine Method

Haversine formula is an equation important in navigation, providing great-circle distances between two points on a sphere from their longitudes and latitudes[2]. The law of haversines, which is relating the sides and angles of spherical triangles, is a special case of a more general formula in spherical trigonometry[3]. Given a unit sphere, a "triangle" on the surface of the sphere is defined by the great circles connecting with three pointsu, **v**, and won the sphere[3]. If the lengths of these three sides are a (from u to v), b (from u to w), and c (from v to w), and the angle of the corner opposite c is C. In Figure 4, shows the illustration of the spherical triangle solved by the law of haversines and the formula for Haversine law. Figure 5, shows the code snippet for the distance calculation based on the Haversine formula.



(i)

$$\text{hav}(c) = \text{hav}(a-b) + \sin(a)\sin(b)\,\text{hav}(C)$$

(ii)

Figure 4. (i) The spherical Triangle solved by Law of Haversines and (ii) Haversine Law.

```
public static float distance(float lat1, float lng1, float lat2, float lng2) {
 double earthRadius = 6371000; //meters
 double dLat = Math.toRadians(lat2 - lat1);
 double dLng = Math.toRadians(lng2 - lng1);
 double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
 Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
 Math.sin(dLng / 2) * Math.sin(dLng / 2);
 double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
 float dist = (float) (earthRadius * c);

 return dist;
}
```

Figure 5. Code Snippet for Distance Calculation based on Haversine Formula.

2.4 Comparison between Distance Calculation Method

The comparison has been made to summarize the features of the three distance calculation methods. Table 1. shows the features comparison between three distance calculation methods.

Based on the Table 1, all the three methods are applicable for Android-based application. distanceTo and distanceBetween methods both are fall under the Google Location Services API whereas the Haversine formula does not belong to any of the API services. From the overall view of this comparison, distanceTo and distanceBetween methods seem much suitable to use compare to the Haversine formula since no limitation has been discovered in these two methods currently.

3. Proposed Work

3.1 General Designs

The HiTracker app is designed to provide users a method to identify the distance between the vehicles or devices. This could benefit the users in case they forgotten where are devices lost. They can track it back by using another device with this HiTracker app installed in it. On the other hand, users can also use HiTracker app to track the nearby vehicles by using devices. This can help the drivers to figure out the distance of the nearby vehicles and minimizes the risk of accident on the road. The setup of the whole HiTracker app is divided into two main areas, which are the backend cloud database and the real time medium.

Table 1. Comparison between Features of Distance Calculation Methods.

Feature	Method		
	distanceTo	**distanceBetween**	**Haversine Formula**
API	Google Location Services API	Google Location API	Not belongs to any API
Applicable for Android?	Yes	Yes	Yes
Function	Returns the distance between the current location and the given location.	Returns the distance between two locations. Optionally computes the final bearings of the shortest path between the two locations.	Calculate the great circle distances between two points on a sphere from their latitudes and longitudes.
Limitation	No	No	Formula does not take into account the non-spheroidal (ellipsoidal) shape of the Earth[4]. It tends to overestimate trans-polar distances and underestimate trans-equatorial distances[4].

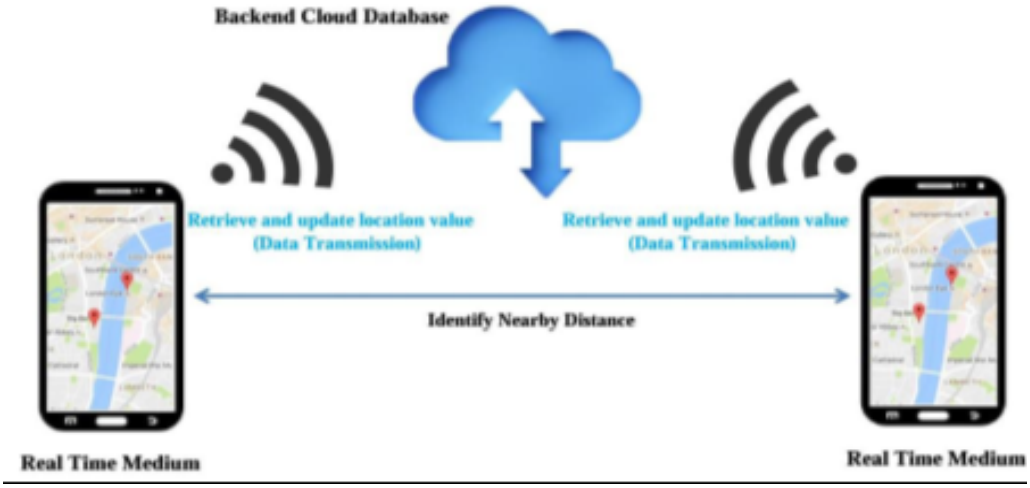


Figure 6. General Concept of HiTracker App.

The main area is where the users will use it (HiTracker app) for identifying the nearby devices on the map by retrieving the current longitude and latitude values. The distance between the nearby devices will be shown in the map and each user will be able to know their respective distance to each other. An alert or the notification will be triggered to warn or remind to that particular, when the nearby devices (another user's device) within a specific range of distance is detected. The most important thing is that users are required to register themselves in order to gain access to the HiTracker app. The account password is able to reset after the registration is done. The second part of the whole HiTracker app is backend cloud database. This is the place where the retrieved location will be stored and updated with the new ones. All the data will be transmitted to the cloud and the synchronization will be taken place where all users (using this app) can tracks their respective location and nearby devices or vehicles at the same time. Figure 6, illustrates the general concept of the whole HiTracker app.

3.2 Geofence

Geo-fencing combine's awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, latitude and longitude are required to be specified. Add a radius on it to adjust the proximity for the location. The latitude, longitude, and radius define a geo-fence, creating a circular area, or fence, around the location of interest[5].

For each geo-fence, Location Services can be requested to send the entrance and exit events, or duration can be specified within the geo-fence area to wait, or dwell, before triggering an event. It is possible to limit the duration of any geo-fence by specifying expiration dura-

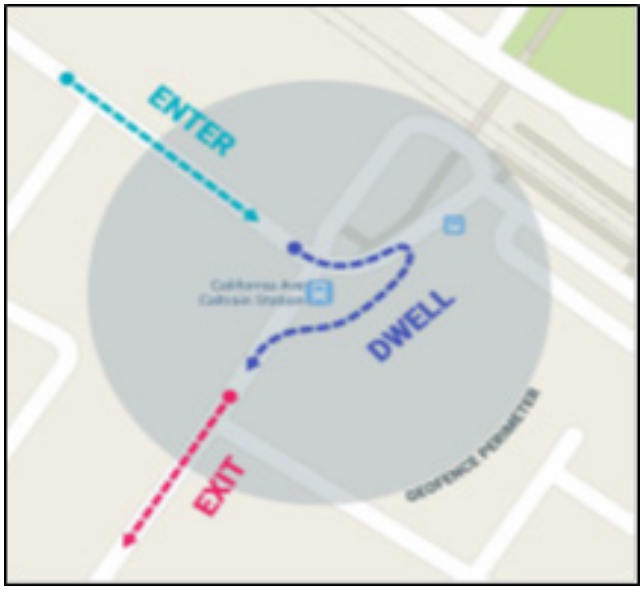


Figure 7. Basic Concept of Geo-Fencing.

tion in milliseconds. After the geo-fence expires, Location Services automatically removes it[6]. Figure 7, shows the basic concept of geo-fencing.

3.3 Distance Calculation Method

Based on the comparison that has been made in findings, distanceTo will be taken as a method to be used in measuring the distance in the HiTracker app. This is due to its flawless feature (no limitation been found currently) of returning the distance from one point to the other point and its ease of use feature. Another reason why distanceTo has been chosen to be apply instead of using distanceBetween is, although distanceBetween has optionally providesthe computation of the final bearings of the shortest path between the two locations, but in order to make the distance calculation works in a simple way, this additional feature of distanceBetween can be ignored (not required in this project).

4. Testing and Results

In HiTracker app, users are required to register an account and login to the user account before they allow to access to the internal content of this app. Also, in case users want to renew their password, they can go to the Settings page of this app. Once they have successfully renew the password, they are required to re-login to the app due to the rules defined in the Firebase Cloud. Other app activities will be further explain in the next few sub-sections. Figure 8, shows the interface of Main Menu page where it will be immediately loaded once the access is permitted and deemed as successful.

Check Devices Activity

Before Check Devices Activity starts, HiTracker app will at first ensure each user has using this app in the presence of internet connectivity. By doing do, the ConnectivityManager class is called in order to answers the queries about the network connectivity state. Besides, it also notifies the app when there are any changes in the network connectivity. To do so, an instance of this class can be obtain or get by just calling getSystemService (Context. ACTIVITY_SERVICE). The core responsibilities of this class are for monitoring the network connections (such as Wi-Fi, GPRS, UMTS and so on), sending broadcast

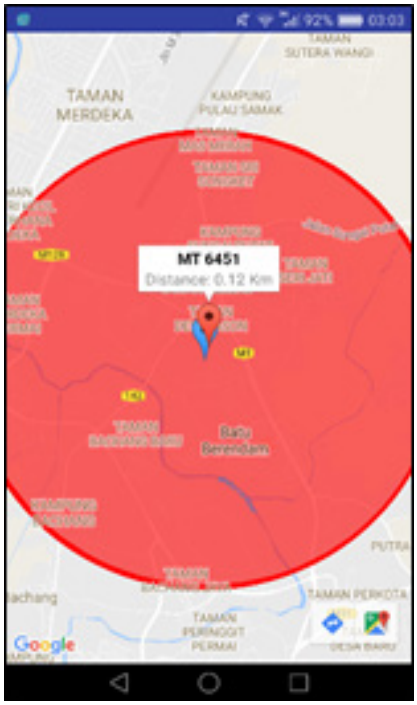
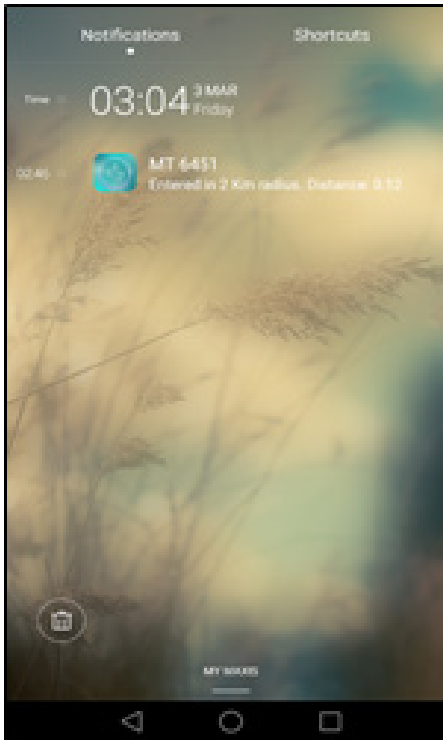


Figure 8. Successful login to the Main Menu page.

(i)



(i)

Figure 9. (i). Distance updating process between the user devices. (ii). The alert received by the user device (with vibration).

intents when there are any changes in network connectivity, attempting to "fail over" to another network when connectivity to a network is lost, providing an API that allows applications to query the coarse-grained or fine-grained state of the available networks and also providing an API that allows applications to request and select networks for their data traffic[7]. Figure 9, shows the code snippet for checking the internet connectivity while opening the app.

Once the user (user device A) clicks the "Check Devices" button, it will redirect user to the map interface where it shows his or her current location with the blue marker stated "My Position". If there is any new user (user device B) comes in within the area of 2 KM from the user device A. The red marker indicates the user device B will show on the map together with the amount of distance (between user device A and user device B). The distance between the two user devices will keep updating for every 60 seconds. User device A will immediately receive an alert (with vibration) when user device B is nearby. Figure 10 (i), shows the map updating the distance between the two user devices (dark red indicates updating process is

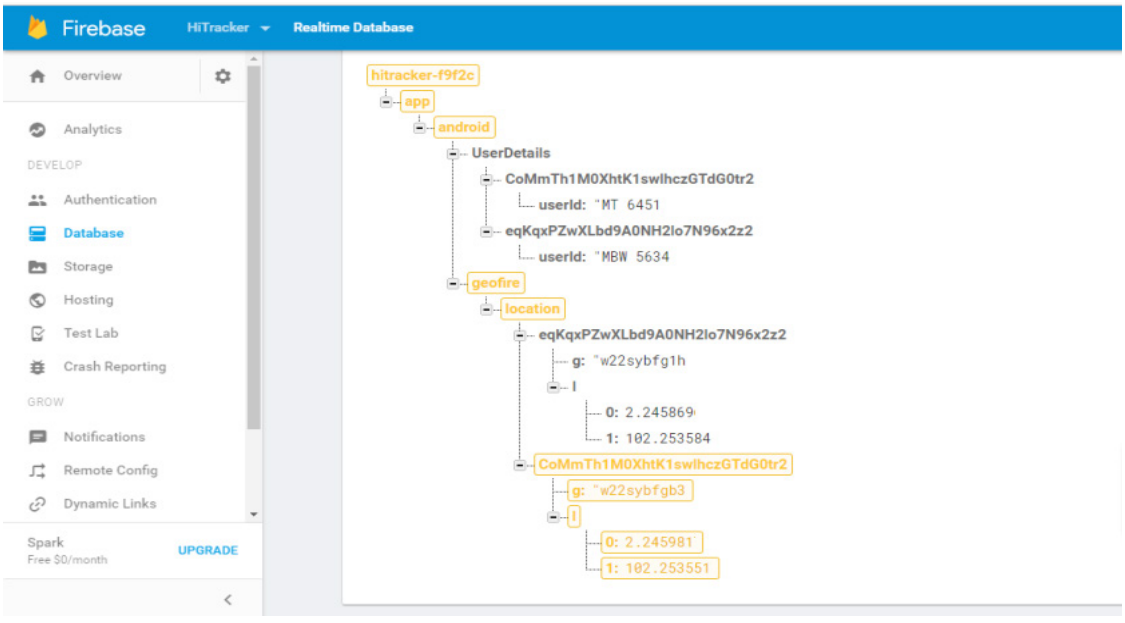


Figure 10. Updating process of the location values for each user devices in Firebase Realtime Database.

running currently) whereas Figure 9 (ii) shows the alert received by the user device (with userId and distance).

Every updated location values (longitude and latitude) collect from the app will then pass to the Firebase realtime database for storage. At the same time, the location values stored in realtime database will be retrieved back by the app from distance notification and measurement. Figure 10, shows the updating process of the location values for each user devices in Firebase Realtime Database.

The last known location of the user device, in other words, user's current location can be requested from the app by using the Google Play services location APIs. An instance of the Google Play services API client (GoogleApiClient.Builder class) need to be created in order to connect to the Location Services API. In order to allows the app to request location or receive permission updates, the device needs to enable the appropriate settings (for example: GPS or Wi-Fi scanning).

Thus, LocationRequest data object is needed to specify the required level of accuracy or power consumption and desired update interval, and the device automatically makes the appropriate changes to the system settings. (Note: setInterval() method sets the rate in milliseconds at which the app prefers to receive location updates. setFastestInterval() method sets the fastest rates in milliseconds at which the app can handle location updates. PRIORITY_HIGH_ACCURACY setting is used to request the most precise location possible and the location services are more likely to use GPS to determine the location). Figure 11, shows the code snippet for connecting to the Google Play services API client and setting up a LocationRequest.

In Figure 11. Code snippet for connecting to the Google Play services API client and setting up a LocationRequest.

Use .getLatitude() and .getLongitude() method in the onLocationChanged to update the location once the changes of the location values is detected. If both method had detected the changes, use updateGeoQuery (in GeoFire) to update the location and store it to the Firebase by using the DatabaseReference. Figure 12, shows the code snippet of updating the changed location values.

To update the marker details (distance) for each user, userLocation is defined to get the lastLocation by using distanceTo() method. Once the distance detail is obtained, it will convert into the required format (in this case, is 2 decimal points and the unit is in KM) and shown it on the

```
googleApiClient = new GoogleApiClient.Builder(this)
        .addApi(LocationServices.API).addConnectionCallbacks(this)
        .addOnConnectionFailedListener(CheckDevice.this).build();
googleApiClient.connect();
locationRequest = LocationRequest.create();
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
locationRequest.setInterval(60 * 1000);
locationRequest.setFastestInterval(10 * 1000);
LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
        .addLocationRequest(locationRequest);
builder.setAlwaysShow(true);
PendingResult<LocationSettingsResult> result = LocationServices.SettingsApi
        .checkLocationSettings(googleApiClient, builder.build());
```

Figure 11. Code snippet of connecting to the Google API.

```
@Override
public void onLocationChanged(Location location) {
    Log.i("Location From Activity", location+" "+googleMap);
    // Update only if location is changed
    if(lastLocation == null || lastLocation.getLatitude() != location.getLatitude() || lastLocation.getLongitude() != lastLocation.getLongitude()) {
        lastLocation = location;
        if (isFirstTime) {
            // isFirstTime = false;
            setGeoQuery(location);
        } else {
            updateGeoQuery(location);
        }
        // Update location at Firebase
        final FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
        DatabaseReference locRef = FirebaseDatabase.getInstance().getReference("/app/android/geofire");
        GeoFire geoFire = new GeoFire(locRef.child("location"));
        geoFire.setLocation(user.getUid(), new GeoLocation(location.getLatitude(), location.getLongitude()), new GeoFire.CompletionListener() {
            @Override
            public void onComplete(String key, DatabaseError error) {
                if(error!=null){
                    Log.i("Error Loc", error.getMessage()+" Details: "+error.getDetails());
                }
            }
        });
    }
}
```

Figure 12. Code snippet of updating the changed location values.

top of the marker. Figure 13, shows the code snippet for updating the marker details for each user.

For generating the notification or an alert, the Notification.Builder() class is used together with some other methods like .setVibrate() (generate vibration), .setContentText() (provides the contents in notification). Later, the NotificationManager class is used to notify the user of events that happen through the retrieval of get-SystemService (Class). (Note: getSystemService is used to return the handle to a system-level service by class).

```
/** Update Marker details for user **/
private void updateMarker(String key , GeoLocation location){
    Marker m = markerDetails.get(key);
    m.setPosition(new LatLng(location.latitude, location.longitude));
    Location userLocation = new Location("GeoFire Provider");
    userLocation.setLatitude(location.latitude);
    userLocation.setLongitude(location.longitude);
    float distance = (userLocation.distanceTo(lastLocation)/1000);
    Log.i("Distance", distance+"");
    String formattedString = String.format("%.02f", distance);
    m.setSnippet("Distance: "+ formattedString+" Km");
}
```

Figure 13. Code snippet for updating the marker details for each user.

```
/** Create Notification and Show it **/
private void createNoti(String user, String Distance) {
    if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.JELLY_BEAN) {
        // Prepare intent which is triggered if the notification is selected
        Intent intent = new Intent(this, CheckDevice.class);
        PendingIntent pIntent = PendingIntent.getActivity(this, (int) System.currentTimeMillis(), intent, 0);
        // Build notification
        Notification noti = new Notification.Builder(this)
                .setVibrate(new long[] { 1000, 1000, 1000, 1000, 1000 })
                .setContentTitle(user)
                .setContentText("Entered in 2 Km radius. Distance: "+Distance)
                .setSmallIcon(getNotificationIcon())
                .setContentIntent(pIntent).build();

        NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        // hide the notification after its selected
        noti.flags |= Notification.FLAG_AUTO_CANCEL;
        notificationManager.notify(0, noti);
    }
}
```

Figure 14. Code snippet for creating and displaying the notification.

Figure 14, shows the code snippet for creating and displaying the notification.

List Devices Activity

Figure 15, illustrates the interface of List Devices in the HiTracker app. Any nearby devices detected within 2 KM from the current shown on the map will be listed in this page. The list will showing the other nearby devices userId and the distance between the current user device. Besides that, the sorting function is provided for the ease of view purpose. The list will be sorted according to the sorting method the app user had chosen.

Figure 15. List Devices interface and the sorting methods.

5. Discussion

There are some limitations can be found in the HiTracker app. HiTracker app does not permit any user to register twice with the same userId during registration. HiTracker app does not provide any checking for identical userId registered an account at the same time. Thus, it might cause the confusion (displaying the app userId on map) when running the "Check Devices" between the user devices. Nevertheless, there are a few advantages where HiTracker app can be applied in daily life. For example, parents can use HiTracker app to identify their children current location in case they want to find out the whereabouts of their children. Another advantage would be, in case any users lost their devices such as mobile phone, they could use HiTracker app in another device to figure

out the place they lost their phone. Hitracker app provides the background service, which allows the app to run at the background even though the current app is closed at the moment. Last but not least, the Hitracker app also provides the live distance monitoring feature which enables each app users to identify and aware of any devices or emergency vehicles approaching to their direction.

6. Results

In the nutshell, the project has been conducted with the aim to identify distance between the nearby devices or vehicles and the suitable cloud services to transmit and retrieve the data, gather all the requirements for HiTracker app, design and develop and assess HiTracker app prototype that can leverage the connectivity (passing location data) in terms of data transmission between the HiTracker app and the backend cloud database and vice versa.

7. Acknowledgement

This project is externally supported by Multimedia University (MMU), Melaka, Malaysia and carried out research and development under the Connected Car Services team.

8. References

1. Location. Available from: https://developer.android.com/reference/android/location/Location.html. Date accessed: 12/12/2016.
2. World Geodetic System 1984. Available from: http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf. Date accessed:
3. The Info List - Haversine Formula. Available from: http://www.theinfolist.com/php/SummaryGet.php?FindGo=haversine_formula. Date accessed: 12/12/2016.
4. Finding distances based on Latitude and Longitude. Available from: http://andrew.hedges.name/experiments/haversine/. Date accessed: 12/12/2016.
5. Receiving Location Updates. Available from: https://developer.android.com/training/location/receive-location-updates.html. Date accessed: 12/12/2016.
6. Creating and Monitoring Geofences. Available from: https://developer.android.com/training/location/geofencing.html. Date accessed: 12/12/2016.
7. ConnectivityManager. Available from: https://developer.android.com/reference/android/net/ConnectivityManager.html. Date accessed: 12/12/2016.