

# Cost and Energy Efficient Real Time Location Tracking Android Application Framework

**Submitted By**  
**Saikat Kumar Sarkar**

**ID: 2012-1-60-016**

Department of Computer Science and Engineering  
East West University

**Supervised By**  
**Dr. Ahmed Wasif Reza**  
**Associate Professor**

Department of Computer Science and Engineering  
East West University



**Dhaka, Bangladesh**  
**August, 2017**

# Declaration

This project has been submitted to the department of Computer Science and Engineering, East West University in the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Engineering by us under the supervision of Dr. Ahmed Wasif Reza, Associate Professor at Department of CSE at East West University under the course 'CSE 497'. We also declare that this thesis has not been submitted elsewhere for the requirement of any degree or any other purposes. This thesis complies with the regulations of this University and meets the accepted standards with respect to originality and quality. We hereby release this thesis to the public. We also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

---

**Saikat Kumar Sarkar**

**ID: 2012-1-60-016**

**Department of Computer Science and Engineering**

**East West University.**

# Letter of Acceptance

The project entitled “Cost and Energy Efficient Real Time Location Tracking Android Application Framework” submitted by Saikat Kumar Sarkar, ID 2012-1-60-016 to the department of Computer Science & Engineering, East West University, Dhaka 1212, Bangladesh is accepted as satisfactory for partial fulfillments for the degree of Bachelor of Science in Computer Science & Engineering in August 2017.

Board of Examiners

1 \_\_\_\_\_

Dr. Ahmed Wasif Reza

Associate Professor

Supervisor

Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

2 \_\_\_\_\_

Dr. Md. Mozammel Huq Azad Khan

Professor and Chairperson

Chairperson

Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

## **Acknowledgements**

First of all, I am grateful to the Almighty God for establishing myself to complete this project. I wish to express my sincere thanks and gratitude to my supervisor Dr. Ahmed Wasif Reza, Associate Professor at Dept. of CSE for the continuous support during our thesis study and related research, for his patience, motivation, and immense knowledge. His guidance helped us in all the time of research and writing of the thesis. I shall always be grateful for having the opportunity to study under him.

I am thankful to all of my teachers, Department of CSE, East West University. I am also grateful to all of my primary and secondary school teachers who were my first teachers in my life and initiator of my basic knowledge.

I would like to express my thanks to my parents for supporting us spiritually throughout writing this project. And I am thankful to all my friends and colleagues. And at last I again thanks to the creator for everything.

## **Abstract**

Object tracking systems plays a vital role in monitoring and surveillance systems. The main target of location tracking systems is to provide object locations at real time with exact accuracy using personalized setup and location sensitivities. Shipping industries at first developed tracking systems to determine ship location at a given time. But those location data used to be stored in local storage which is accessible only when the ship is arrived. But with the flow of time and advancement of technology, now we want to know object location at real time. As the local storage cannot provide real time data, we moved to server based systems for real time data parsing. Today we have smart devices like android phones that can detect location data by itself. But to share and monitor those data we need an online system. Here comes the server based location tracking system. But many factors like battery consumption, data load cost, accuracy of data make this system a bit complex. We have tried to solve these problems and improve existing systems by proposing an android based framework that uses cloud services for real time data storage and processing and firebase cloud messaging for device to device communication at real time. We also propose a cost and energy efficient framework for android devices for location tracking and data transmitting.

# TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	<b>List of Figure</b>	<b>vii</b>
	<b>List of Abbreviations</b>	<b>viii</b>
<b>1</b>	<b>Introduction</b>	<b>01-04</b>
1.1	Background	01
1.2	Problem Statements	02
1.3	Research objective	02-03
1.4	Thesis contribution	03
1.5	Thesis Organization	03-04
<b>2</b>	<b>Existing System Review</b>	<b>05-08</b>
2.1	Survey of Existing Systems	05-08
2.2	Summary	08
<b>3</b>	<b>Project Methodology</b>	<b>09-23</b>
3.1	Propose System	09-15
3.2	Algorithm and Pseudocodes	15-20
3.3	Performance Optimization	20-23
3.4	Implementation Sectors	23
3.5	Summery	23

<b>4</b>	<b>Results</b>	<b>24-25</b>
<b>5</b>	<b>Conclusion</b>	<b>26</b>
5.1	Overall Conclusion	26
5.2	Future works	26
	<b>References</b>	<b>27</b>

## LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
2.1	Existing location tracking architecture -I	06
2.2	Existing location tracking architecture -II	07
3.1	General architecture the proposed system	09
3.2	Overall architecture of proposed system	10
3.3	Location tracking architecture	11
3.4	Friend list processing architecture	12
3.5	Permissions accessed by application	13
3.6	Dependencies of application	13
3.7	User Registration Process	14
3.8	User Login Process	15
3.9	User Authentication database snapshot	15
3.10	Friend Request table snapshot	16
3.11	Friend Request Acceptance table snapshot	17
3.12	Location table snapshot	18
3.13	User table snapshot	19
3.14	Notification table snapshot	19
3.15	Cloud Function Console	20
3.16:	Location accuracy constants	21
4.1	Application Map Screenshot	24
4.2	User online/ offline	25



## **LIST OF ABBREVIATIONS**

GPS	-	Global Positioning System
GCM	-	Google Cloud Messaging
FCM	-	Firebase Cloud Messaging
JSON	-	JavaScript Object Notation
API	-	Application programming interface
REST	-	Machine learning
AI	-	Artificial Intelligence
IoT	-	Internet of Things

# CHAPTER 1

## INTRODUCTION

### 1.1 BACKGROUND

Now a day's real time location sharing systems have a huge impact on object tracking and monitoring services. Tracking an object location which is moving and changing its location continuously is not an easy job. Tracking these type of objects at real time makes the job even harder. The main challenge is to collect location data at runtime. Manual input system does not provide expected result.

But in the era of IoT, smart devices and cloud services this job is possible now. The android devices have GPS and location sensors at can collect device location data. Many location broadcasting system uses android devices to get user location. So, getting user location at runtime without manual input is not hard now. But sharing that location at real time increases the complexity.

At real time data transfer the main challenge lies in data receiving and transmitting speed of server side. In location sharing system on android devices the second complexity is location detection with less battery drain. The third complexity is to minimize the load without compromising data quality.

In our proposed system we have tried to solve these complexities using cloud services, asynchronous battery power monitoring system and effective data shrinking system.

## **1.2 PROBLEM STATEMENTS**

We are going to propose an android application based real time object location sharing system that drains less battery power, uses less data to share location and depends on only one backend cloud server for storage and messaging.

Observing the existing systems, we have noticed several limitations like almost all systems use GPS (Global Positioning System) and positioning sensors to locate the position of an object. But the accuracy GPS is questionable if the object resides in a building, surrounded by trees or there is an extreme atmospheric conditions such as geomagnetic storms. The GPS based location tracking systems also drains large amount of battery power of android devices. As real time location sharing demands high amount of data transfer, the cost of data usages is a bit high.

Another drawback of many androids based real time location sharing systems is the dependency on two different backend server, one backend app server for data storage and logic implementations and another one cloud server for real time data sharing.

So, we have tried to increase the accuracy of the object locations detection and provide efficient battery power management and cost optimization technique.

## **1.3 PROJECT OBJECTIVES**

- I. To develop a real time location sharing android application that detects objects locations with more accuracy.

- II. To develop an efficient battery power consumption method for real time location sharing android applications.
- III. To decrease installation cost and data transfer rate

#### **1.4 PROJECT CONTRIBUTIONS**

- I. We developed a real time location sharing android application that works in real life.
- II. We have introduced a better location detection technique using Google API Client and Fused Location API.
- III. We have developed an AI that reads the battery percentage and chooses better location detection plan at runtime
- IV. We have developed a moderate method of JSON data parsing for FIREBASE cloud platform that minimizes that data load.
- V. We have introduced a cloud based single server management system as the backend of the application.

#### **1.4PROJECT ORGANIZATION**

The following is an overview of the contents of the chapter that presented in this research:

**Chapter 2:** Chapter 2 provides an overview of the literature survey on existing real time location sharing systems, location detection techniques, system installation process and data transfer techniques.

**Chapter 3:** On chapter 3 we will discuss our proposed system architecture, describe the

proposed system, how it works and also give the algorithm and techniques related to this project.

**Chapter 4:** On chapter 4 we will show the results of our android application.

**Chapter 5** Chapter 5 is the concluding chapter that describes the summary of this thesis which visualization by analysis and we also provide some recommendations for further research and future works.

## CHAPTER 2

### EXISTING SYSTEM REVIEW

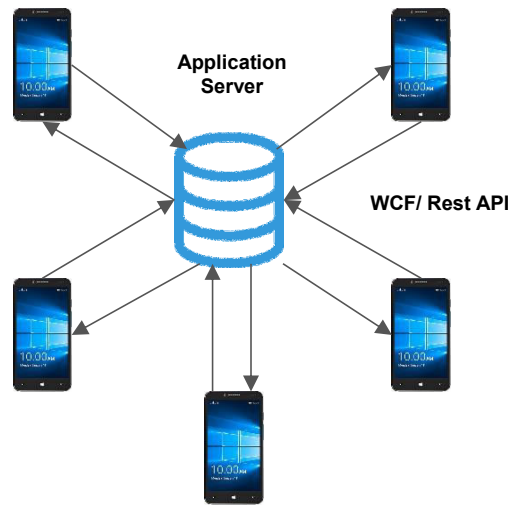
#### 2.1 SURVEY OF EXISTING SYSTEMS

We can find a different type of implementations for real time location sharing android based applications. Before describing in details we can concentrate on following point about these systems,

- I. Uses **Android's Location API** to get object/ user location. The location API uses three type of provider to get location and they are,
  - a) **GPS\_PROVIDER**: This provider determines location using satellites.
  - b) **NETWORK\_PROVIDER**: This provider determines location based on availability of cell tower and WiFi access points.
  - c) **PASSIVE\_PROVIDER**: This provider will return locations generated by other providers.
- II. Uses a windows or Linux based backend servers to store, process and transmit data.
- III. Mostly uses JSON format data for messaging

In our survey, we have found mainly two types of architecture. One of them uses single backend server for storing, processing and transmitting data. The another one uses a cloud server for messaging with the backend server for storage and processing. Another architecture which is recently introduced is single cloud based backend server.

## Single Backend Server:



**Figure – 2.1: General architecture of single server based android application system**

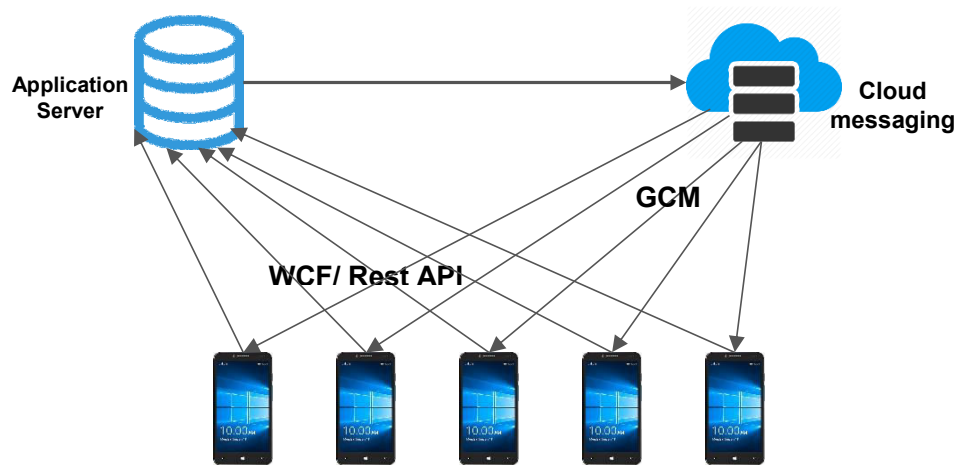
In this architecture the android devices collect location data using **Location API** in a background service and send to a local server in JSON format using Rest API or WCF services. The data is stored in a table including the user key. In a different table connection between all the users (Friend list) is stored. In another background service the android devices continuously queries about any change on their friend's location table and on every location update the server transmits the location data to the user's friends list. The devices receive the updated location data and put a marker on the map.

The main drawback of this system is,

- i. Sending and receiving data using Rest API is a bit slow. It requires 10-20 seconds, in worst case 30 seconds to send and receive user location.
- ii. The drawback of JSON format data is that though it is a fast data structure as it formats data in key value pair, only value part of the data is usable, the key part of the data is an extra whose role is to notify the actual point of the value.
- iii. Excessive traffic at real time can overwhelm the server as it has to receive, process, transmit the data. The server also needs extra bandwidth to provide real time

service.

### Backend Server with Cloud Server:



**Figure – 2.2: General architecture of backend Server with Cloud Server android application system**

In this architecture, like the previous one devices sends data to a server. The server is used for storing, processing and logic implementation on data. Then instead of sending the data directly to devices it sends the data to a cloud server in a bundle where each device is registered with a unique key. Then the cloud server sends the data to corresponding devices. Though sending data to a server is still slow but receiving data from cloud service is a bit faster.

In reference [1], the authors proposed a system that collects location data using GPS and stores it in local storage. It shows how the location data can be used in app location tracking. In reference [2] the authors used backend server with php based web application as admin panel to store user location and alert service. The system checks user location at real time and if user enters into or exits from a restricted area, an alarm will be triggered.



In reference [3], the authors proposed system is same as described in Figure: 2.2. The proposed system in reference [4] uses only cloud server to store vehicle location and tracking. But the system does not care about power and cost optimization. In reference [5] the system uses local server and cloud server for real time vehicle tracking using GPS.

## **2.2 SUMMARY**

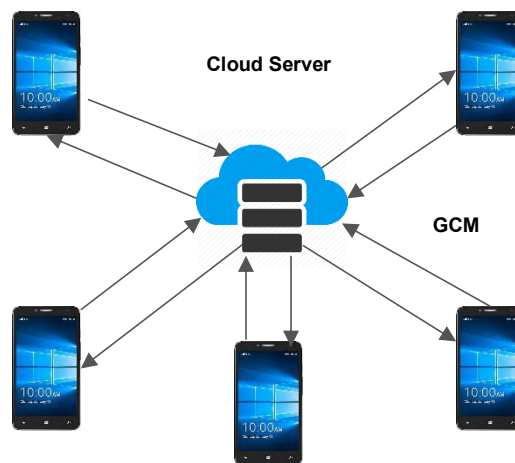
The part of literature review has clearly brought about the fundamental overview of the existing location sharing systems, their architectures and drawbacks. There are many challenges in implementing real time location sharing system like managing device battery power consumption, decreasing data transfer rate or managing server side and increasing data transfer speed. Our proposed system aims to provide a power and cost efficient android mobile application architecture that depend only on cloud server.

## CHAPTER 3

### PROJECT METHODOLOGY

#### 3.1 PROPOSE SYSTEM

No Backend Server, only Cloud Server:



**Figure – 3.1: General architecture of our proposed system**

In our proposed system we used only cloud based server for data storage, user authentication, file sharing, location sharing and push messaging. It is a *IoT based system* where at first user himself/ herself to the system using firebase cloud authentication. User can use firebase cloud storage to store image, audio, video file. User can post single line status. A background service independent of the application automatically collects and sends the location data to server continuously maintaining a fixed interval. This interval changes with the device battery percentage. A user can have only one account using an email address. Once user

is logged in he/ she can share location, check which of his friends are online, make a friend list, chat with them, check which of his friend is nearby on google map.

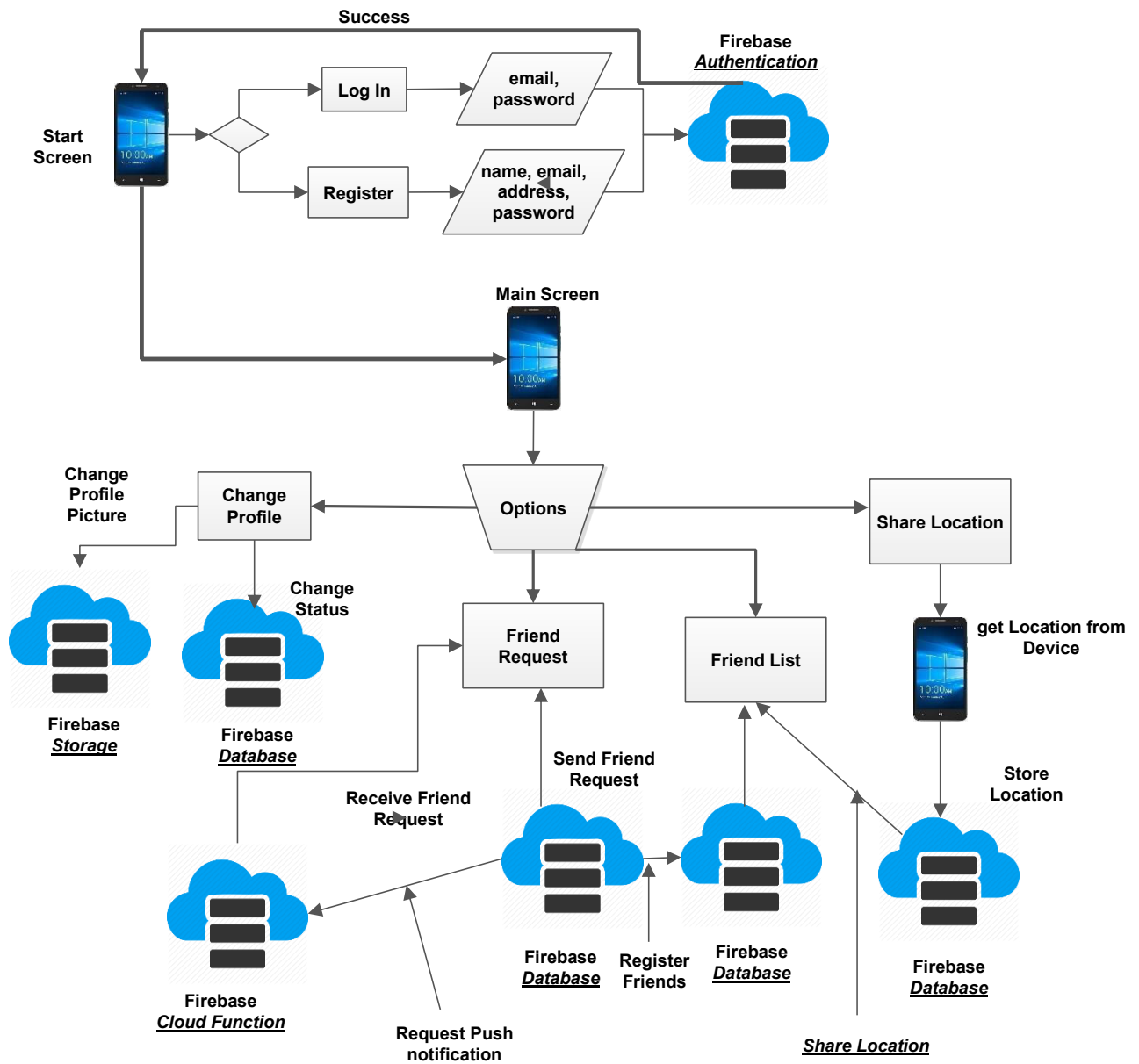
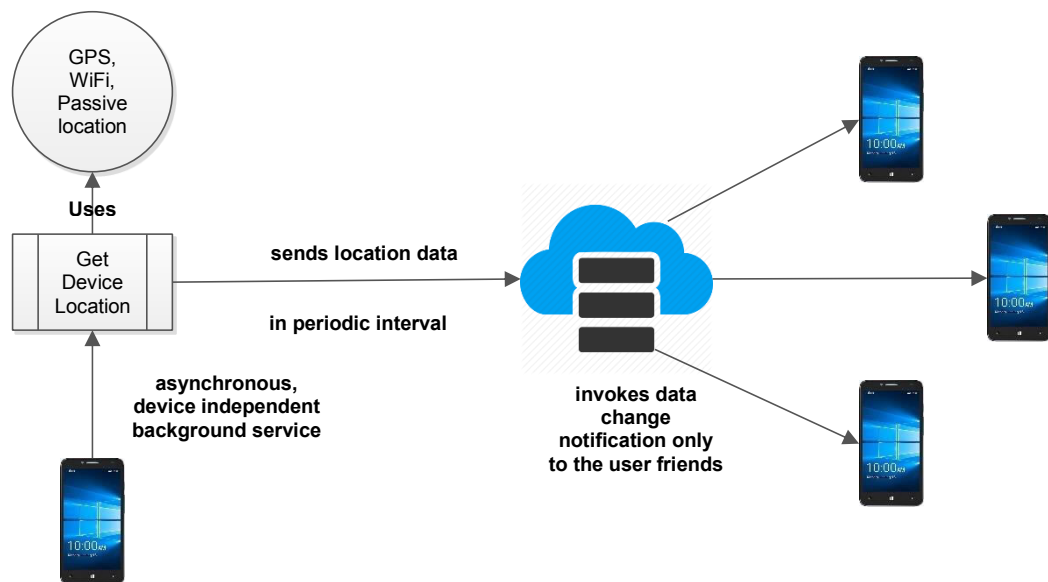


Figure – 3.2: Overall architecture of our proposed system

**IoT Based Location Tracking:** The location tracking method of our proposed system is device independent. A lot of existing system proposes that device has to let the application running to get current location. But in our case if the user allows the application to collect location data, it will start a background service which independent of application runtime. Once the background location service starts, it will collect the location data in predefined interval and push the data to cloud database. If friends of our current user are watching him on map, then on each data change the server will invoke a data change notification and the data change listener on friend's device will use that notification to get location update and put a marker on that location.



**Figure – 3.3: Location tracking architecture of our proposed system**

**IoT Based Friend Request:** Our friend request processing is also IoT based. The current

user basically selects a user from the user list to send friend request. The integrated firebase cloud agent in application collects the requesting and requested user id and sends it to cloud server. The server generates a push notification for the requested user and sends it. The firebase agent in requested user's device receives the friend request and shows it as notification. If the requested user accepts the friend request, their state changes to "friends".

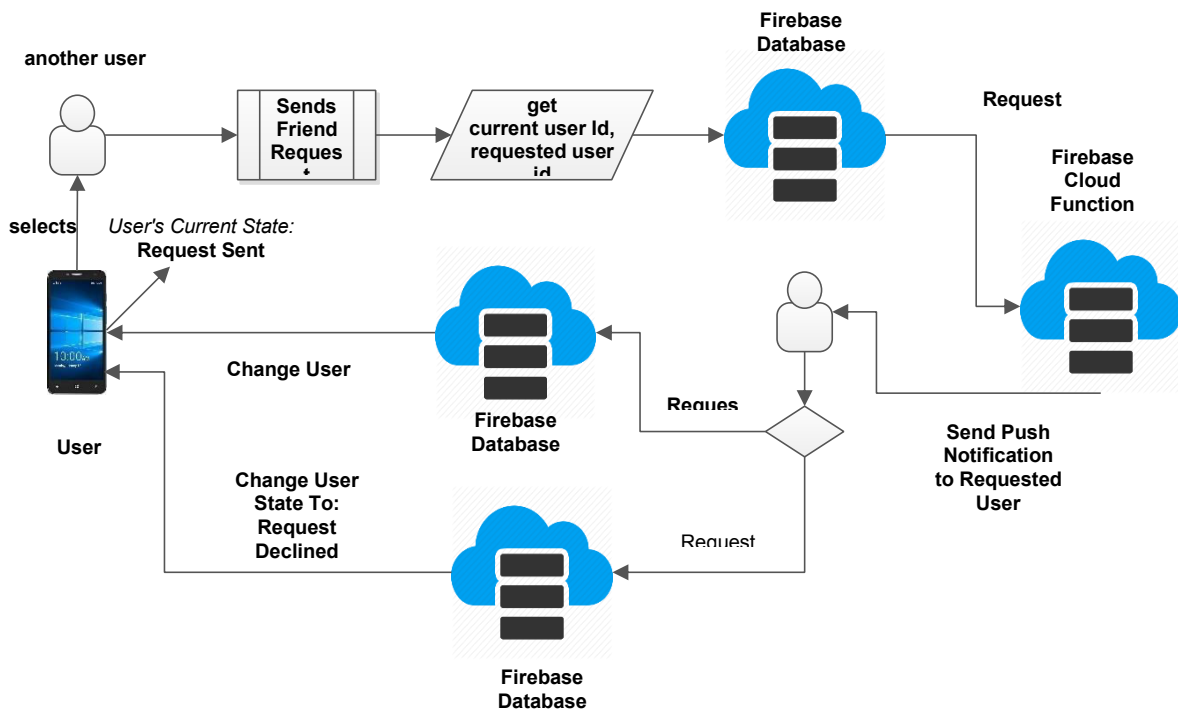


Figure – 3.4: Friend requesting and accepting architecture of our proposed system.

**Use Cases:**

- a) User registration and one-time login, user will not need to login again and again unless he/ she logs out.
- b) User authentication is managed in backend using cloud authentication on email and password.
- c) User can share his/ her current status and it will be shown at real time on his/ her

profile.

- d) User can send friend request, receive them, unfriend existing friend and decline friend request.
- e) If someone sends friend request to the user, he/ she will be notified by a push notification.
- f) User can share his/ her current location among friends.
- g) User can watch his/ her friend's current location and full path at real time.
- h) User can use the one to one real time chat system for messaging.
- i) User can send danger alert to his friends in case of emergency and ask for help.
- j) It will be shown at real time weather the user is offline or online.

#### Permissions:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />  
<uses-permission android:name="android.permission.READ_PROFILE" />  
<uses-permission android:name="android.permission.READ_CONTACTS" />  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure – 3.5: Permissions accessed by the application

#### Application Level Dependencies:

```
compile 'com.google.firebase:firebase-auth:10.0.1'  
compile 'com.google.firebase:firebase-auth:10.0.1'  
compile 'com.android.support:appcompat-v7:25.3.1'  
compile 'com.android.support.constraint:constraint-layout:1.0.2'  
compile 'com.android.support:design:25.3.1'  
compile 'com.google.firebase:firebase-database:10.0.1'  
compile 'com.android.support:support-v4:25.3.1'  
compile 'de.hdodenhof:circleimageview:2.1.0'  
compile 'com.theartofdev.edmodo:android-image-cropper:2.4.7'  
compile 'com.google.firebase:firebase-storage:10.0.1'  
compile 'com.squareup.picasso:picasso:2.5.2'  
compile 'com.squareup.okhttp:okhttp:2.5.0'  
compile 'com.firebaseui:firebase-ui-database:1.1.1'  
compile 'id.zelory:compressor:2.1.0'  
compile 'com.google.firebase:firebase-messaging:10.0.1'  
compile 'com.google.android.gms:play-services-maps:10.0.1'  
compile 'com.google.android.gms:play-services-gcm:10.0.1'  
compile 'com.google.android.gms:play-services-location:10.0.1'
```

Figure – 3.6: The Application level dependencies

**Gradle Level Dependency:**

```
compileSdkVersion 25
buildToolsVersion "25.0.2"
minSdkVersion 21
targetSdkVersion 25
classpath 'com.google.gms:google-services:3.0.0'
plugin: 'com.google.gms.google-services'
```

**Used Services:**

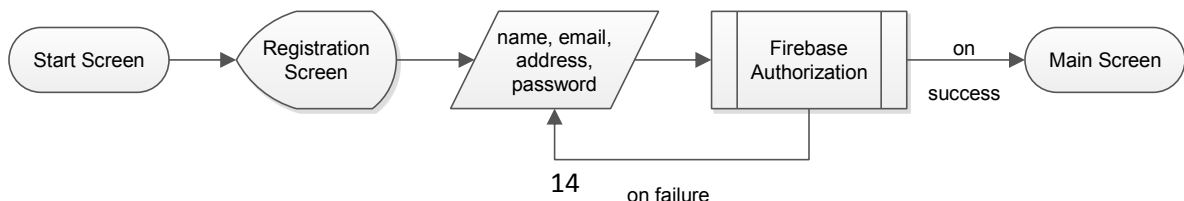
- i. Google Play services
- ii. FCM (Firebase Cloud Messaging) services.
- iii. Firebase cloud functions
- iv. Firebase Realtime Database
- v. Firebase Authentication
- vi. Firebase Storage
- vii. Google API Client
- viii. Google Maps API

Allowed Android OS: Version 21 and above

Backend Server: Firebase

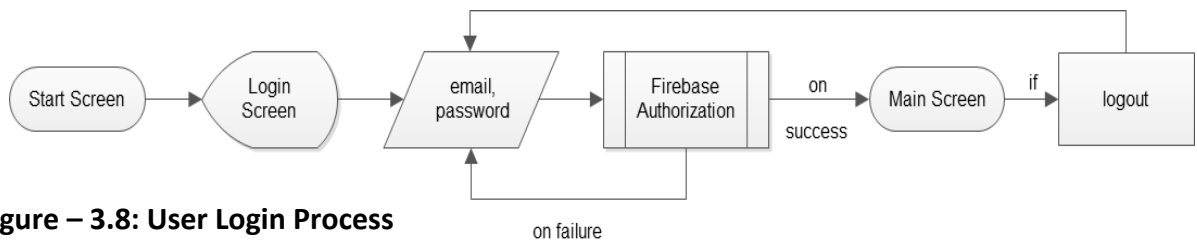
**User Authentication:**

Registration: In case of registering to the system user has to provide name, email address, local address/ locality and password. Once user is registered into the system a passive user id will be generated and this id will always be used to identify user and access backend.



**Figure – 3.7: User Registration Process**

Login: User has to provide email and password to login. Once the user is logged in, it is not necessary to login every time unless user is logged out. The firebase authentication system provides the user id which is synced with a device token that matches the user authenticity.



**Figure – 3.8: User Login Process**

Database Snapshot:

<input type="text" value="Search by email address, phone number, or user UID"/> <span style="float: right;"> <a href="#">ADD USER</a> <span style="margin-left: 10px;">↻</span> </span>				
Identifier	Providers	Created	Signed In	User UID
saikat@gmail.com	✉	Aug 4, 2017	Aug 4, 2017	j4Qlz9m37z0m8FThPbSnePNSG1...
rakib@gmail.com	✉	Jul 31, 2017	Jul 31, 2017	zFPHcFUP0aPy6PJA5KlqCVL4DYJ2
pasha@gmail.com	✉	Jul 31, 2017	Aug 4, 2017	zTVYcN3tsqdPRJ5gFepMT6aJ7va2

**Figure – 3.9: User Authentication database snapshot**

### 3.2 ALGORITHM AND PSEUDOCODES

**Friend Request Process:** User generally selects a user profile to send friend request. The database receives the request and checks that if there is any “friend\_request” table. If no table exists, it creates a table and two nodes under that table. In one node under the requested user key it creates new node with the requesting user key. In another node it does the opposite.



Then Firebase uses cloud functions to send the requested user a notification about the friend request.

```
Get user id from current user context
If(friend_requese_table != null){
    Find user id node;
    Create a child node using requested user id
    Create a child node with key "request type"
    Add value to key "sent"
    If(requested user id exists in table){
        Find that node
        Create a child node using requesting user id
        Create a child node with key "request type"
        Add value to key "received"
    } else {
        Add a node using requested user id
        Create a child node using requesting user id
        Create a child node with key "request type"
        Add value to key "received"
    }
} else {
    Create friend_requese_table
    Do above process
}
```



**Figure – 3.10: Friend Request table snapshot**

**Friend Request Accepting Process:** When user gets a friend request by notification, he can accept the request or decline it. If user declines the request the nodes under friend request table are deleted. If user accepts the friend request, then also the nodes under friend request table is deleted and new table is created named “friends”.

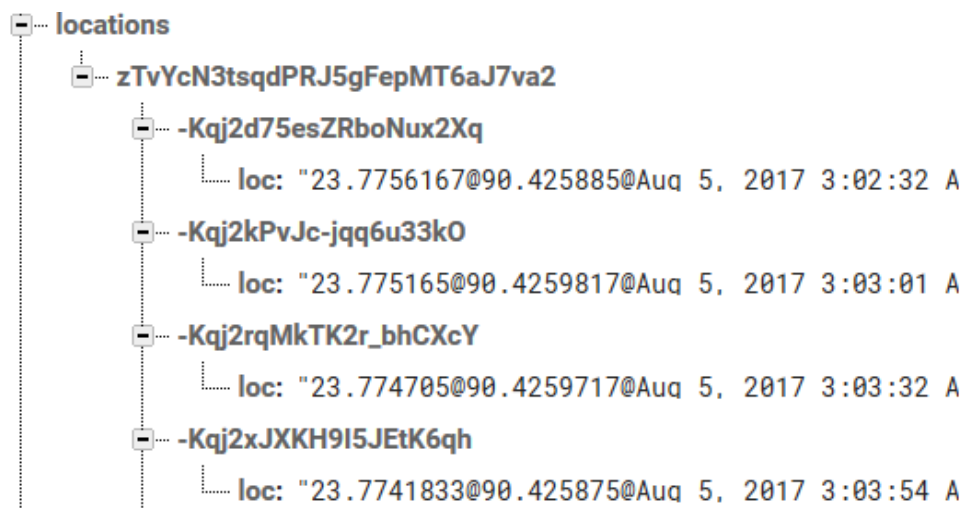
```
Get user id from current user context
If(friends_table != null){
    Find user id node;
    Create a child node using requested user id
    Create a child node with key “date”
    Add value to key “device date and time”
    If(requested user id exists in table){
        Find that node
        Create a child node using requesting user id
        Create a child node with key “date”
        Add value to key “device date and time”
    } else {
        Add a node using requested user id
        Create a child node using requesting user id
        Create a child node with key “date”
        Add value to key “device date and time”
    }
} else {
    Create friends_table
    Do above process
}
```



**Figure – 3.11: Friend Request Acceptance table snapshot**

**Getting Location Data and sending to server:**

1. Used call back methods: GoogleApiClient.ConnectionCallbacks, LocationListener  
GoogleApiClient.OnConnectionFailedListener
2. Constants:
  - Update Interval: After each interval location data will be retrieved
  - Fastest Interval: On position changes of devices no update will be taken before this interval
  - Priority: It will determine the accuracy level of location data
  - Smallest displacement: location will be updated if device crosses the displacement distance.
3. API Used: LocationServices.FusedLocationApi
4. Related Methods:
  - onConnected → System listens to location update
  - onConnectionSuspended → request to reestablish google API client connection
  - onConnectionFailed → reports connection failure status
  - onLocationChanged → returns current Latitude, Longitude, accuracy etc
5. A background service retrieves the location data, then pushes it to server periodically specified by the constants in JSON format.



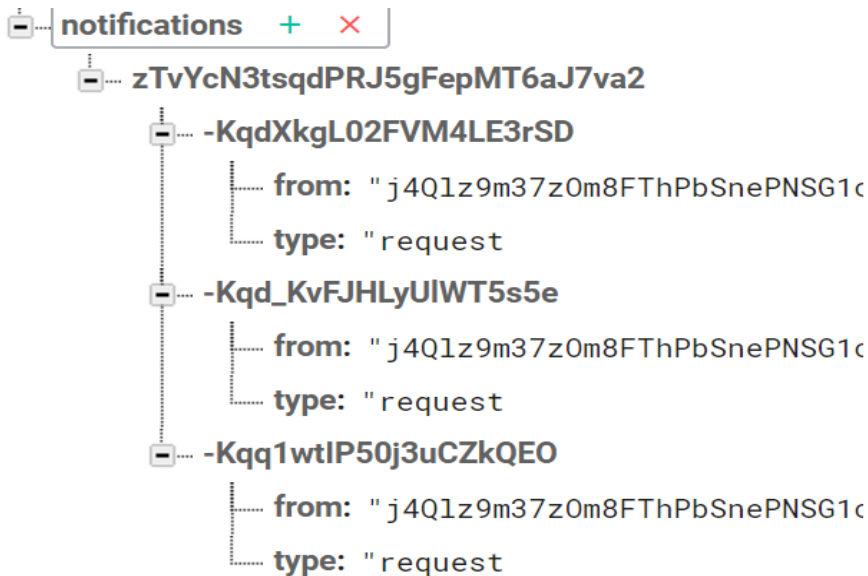
**Figure – 3.12: Location table snapshot**

**Database Structure:** User table contains user data in JSON format. It mainly saves user name, address, email address, status, online status, unique device token etc.



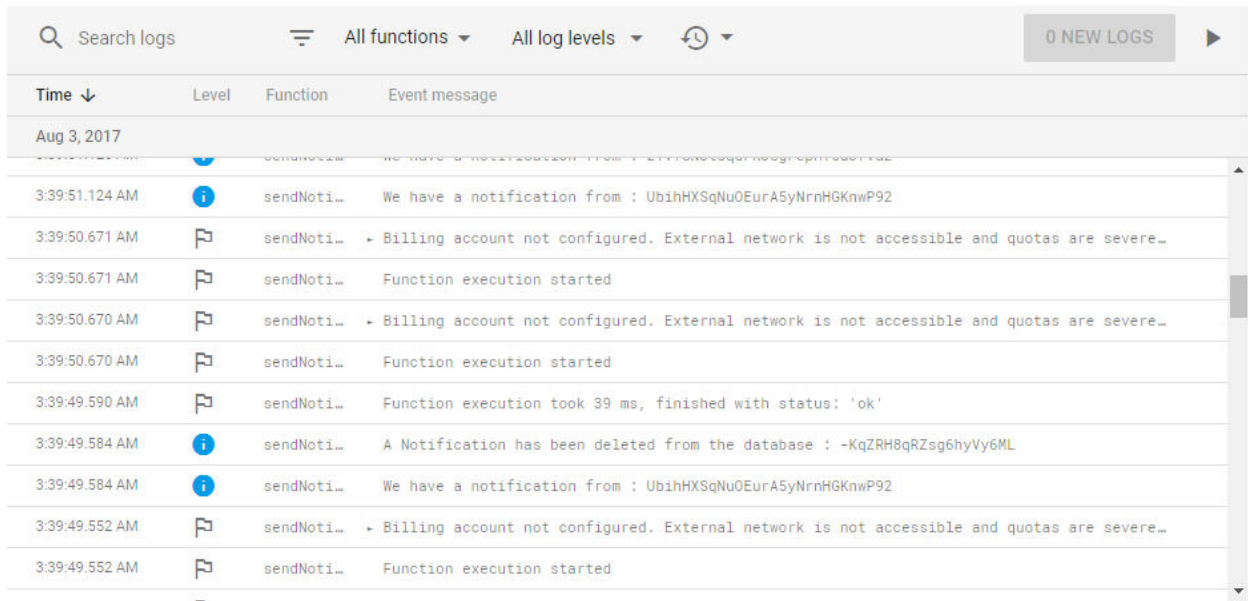
**Figure – 3.13: User table snapshot**

Notification table contains the sent messages and the device Ids.



**Figure – 3.14: Notification table snapshot**

**Cloud Function:** The cloud function mainly adds the internal logic on database. Here we are using cloud function to find out a user a profile, device token and send push notification.



Time ↓	Level	Function	Event message
Aug 3, 2017			
3:39:51.124 AM	<span style="color: blue;">i</span>	sendNoti...	We have a notification from : UbihHXSqNu0EurA5yNrnHGKnwP92
3:39:50.671 AM	<span style="color: red;">!</span>	sendNoti...	↳ Billing account not configured. External network is not accessible and quotas are severe...
3:39:50.671 AM	<span style="color: red;">!</span>	sendNoti...	Function execution started
3:39:50.670 AM	<span style="color: red;">!</span>	sendNoti...	↳ Billing account not configured. External network is not accessible and quotas are severe...
3:39:50.670 AM	<span style="color: red;">!</span>	sendNoti...	Function execution started
3:39:49.590 AM	<span style="color: red;">!</span>	sendNoti...	Function execution took 39 ms, finished with status: 'ok'
3:39:49.584 AM	<span style="color: blue;">i</span>	sendNoti...	A Notification has been deleted from the database : -KqZRH8qRZsg6hyVy6ML
3:39:49.584 AM	<span style="color: blue;">i</span>	sendNoti...	We have a notification from : UbihHXSqNu0EurA5yNrnHGKnwP92
3:39:49.552 AM	<span style="color: red;">!</span>	sendNoti...	↳ Billing account not configured. External network is not accessible and quotas are severe...
3:39:49.552 AM	<span style="color: red;">!</span>	sendNoti...	Function execution started

**Figure – 3.15: Cloud Function Console**

### 3.3 PERFORMANCE OPTIMIZATIONS

#### Backend Optimizations:

As we have discussed earlier in literature review the existing systems uses local windows/ Linux based server to store, processing and transmit data, we can notice that a backend application is also needed to implement logics on data. It generally requires 10 seconds to twenty seconds to complete a send and receive circle form android device which is very low performance for real time communication.

As real time communication needs huge data flow, local database can be overwhelmed with data flow. Using cloud massaging service though decreases the send and receive circle time to 10 -12 seconds, still it is not fast enough.

Our system uses FIREBASE cloud database and hosting service provided by google. It does not use any local storage to store and process user data. It authenticates users using firebase authentication. Stores user data in firebase database. Stores user uploaded files in firebase storage implements logic on data using firebase cloud function.

The average completion time of a read write circle is 0.5-3 seconds. It does not need any Rest API call for requesting data. Firebase can easily be integrated with android devices. As it is a cloud service huge data processing is way faster than local servers.

**Battery Power Consumption Optimization:**

One of the main drawback of android based location sharing system is that as GPS highly depends on battery power, the power drain for sharing location is very high. So we developed backend service system that continuously checks the battery level of the device and chooses suitable location sharing options read battery status data.

Here we need to describe the constant values. At first we can take a look at this constants description provided by google.

Constant Summary

int	<code>PRIORITY_BALANCED_POWER_ACCURACY</code>	Used with <code>setPriority(int)</code> to request "block" level accuracy.
int	<code>PRIORITY_HIGH_ACCURACY</code>	Used with <code>setPriority(int)</code> to request the most accurate locations available.
int	<code>PRIORITY_LOW_POWER</code>	Used with <code>setPriority(int)</code> to request "city" level accuracy.
int	<code>PRIORITY_NO_POWER</code>	Used with <code>setPriority(int)</code> to request the best accuracy possible with zero additional power consumption.

**Figure – 3.16: Google provided location accuracy constants**

So we have created an algorithm that reads the battery percentage in an asynchronous service and chooses following plans,

- If battery power is greater than 70%
  - Priority → HIGH\_ACCURACY
  - Update Interval → 10 seconds
  - Fastest Interval → 5 seconds
  - Displacement → 10 meter
  
- If battery power is between 70% and 55%
  - Priority → HIGH\_ACCURACY
  - Update Interval → 15 seconds
  - Fastest Interval → 10 seconds
  - Displacement → 10 meter

- If battery power is between 55% and 30%
  - Priority → BALANCED\_POWER\_ACCURACY
  - Update Interval → 15 seconds
  - Fastest Interval → 10 seconds
  - Displacement → 15 meter
  
- If battery power is between 30% and 15%
  - Priority → BALANCED\_POWER\_ACCURACY
  - Update Interval → 20 seconds
  - Fastest Interval → 15 seconds
  - Displacement → 20 meter

### Decreasing Data Transfer Load:

Generally android applications communicate with server using JSON format data. The JSON data is a key value paring data set. Let us take a look at a JSON set,

```
{
  "user_name": "Jhon Doe"

  "latitude": "2.758486"

  "longititude": "3.758496"

  "status": "online"

  "device_token": 8xcfsfgg

  "date" : 3.36 PM 2nd August
}
```

Here we can notice that we only need "Jhon Doe, 2.758486, 3.758496, online, 8xcfsfgg , 3.36 PM 2<sup>nd</sup> August" these data to process user location. But we also sending and receiving "user\_name, latitude, longitude, status, device\_token, date" these keys to find out the data description. If each character is 1 byte then, here we send and receive

Total =137 character \* 1 byte = 137 byte  
 Where,  
 Data = 69 character \* 1 byte = 69 byte  
 Key = 68 character\* 1 byte = 68 byte

So, if system updates location data in every 10 seconds and user uses real time location service for 5 hours,

Then, total data send or receive =  $137 * 6 * 60 * 5 = 246600$  byte = 1.9728 mb

Where usable data =  $69 * 6 * 60 * 5 = 124200$  byte = 0.9936 mb

And data for key =  $68 * 6 * 60 * 5 = 122400$  byte = 0.9792mb

In more general cases we observed that in JSON data parsing we send/ receive 20 -35% extra data as key.

Firebase also stores data in JSON Format, but we have moderated the storage method.

As real time system accesses data very frequently, adding extra key data can cause huge data cost. So we have formatted the in a fixed order and format.

Order: name → latitude → longitude → status → device\_token → date

Data → Jhon Doe → 2.456789 → 2.345672 → online → 85fbidls → 3.45 pm 2<sup>nd</sup> August

Formatted data:

{loc\_data: Jhon Doe @ 2.456789 @ 2.345672 @ online @ 85fbidls @ 3.45 pm 2<sup>nd</sup> August }

We are separating data using “@” sign. As the data is generated by device itself, no user interaction is involved and the condition is to send all the field or do not send anything, there is very less possibilities of order violation. At receiver end we split the string by “@” sign to find desired data.

### 3.4 IMPLEMENTATION SECTORS

- Child and adult citizen monitoring
- Sharing location for seeking help in danger
- Sales service tracking
- Lost device tracking
- Biker race monitoring
- Creating Friend Jones
- Product delivery service tracking.

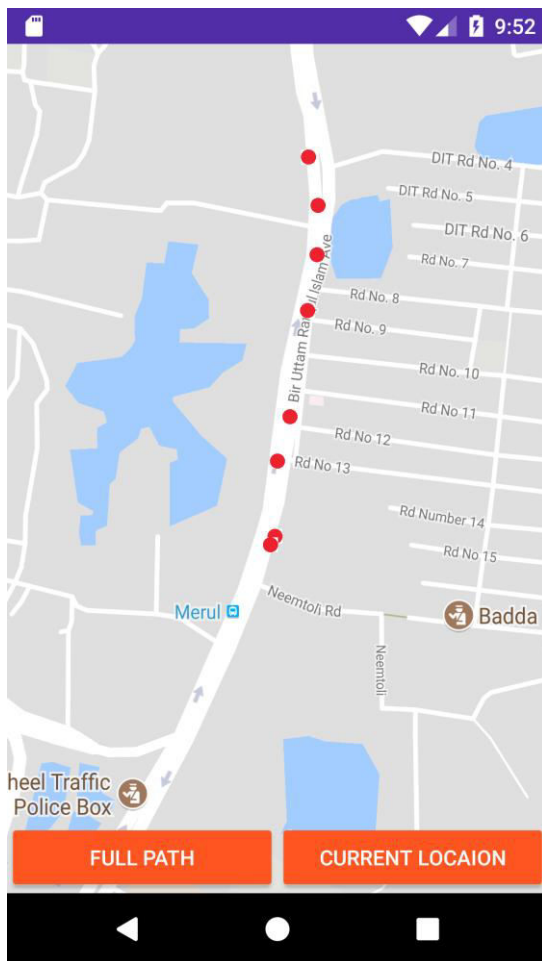
### 3.5 SUMMERY:

In this project we have developed an android application framework that collects location data at real time and sends to cloud database using the optimized procedures described at chapter 3.3. It shares location at real time and other users can watch and monitor user’s movement on google map.

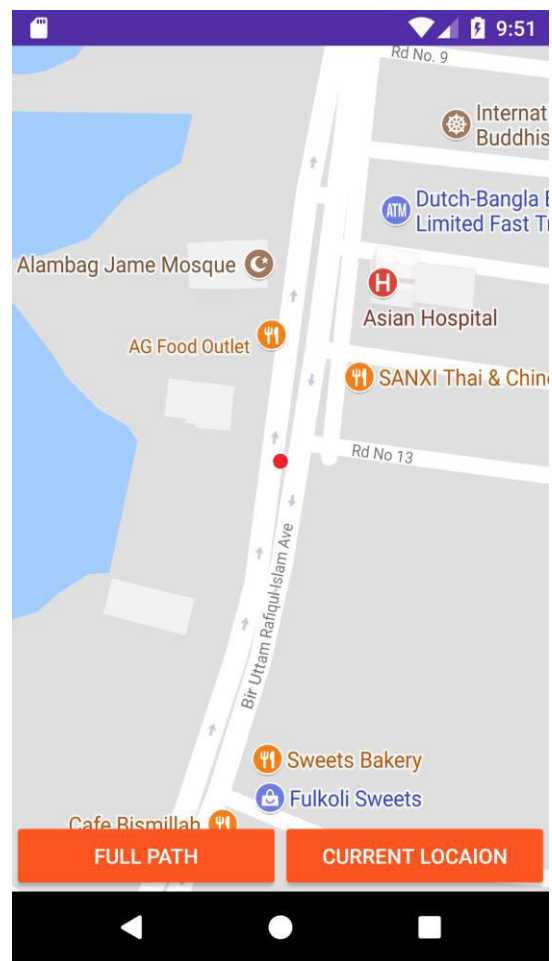


# CHAPTER 4

## RESULTS

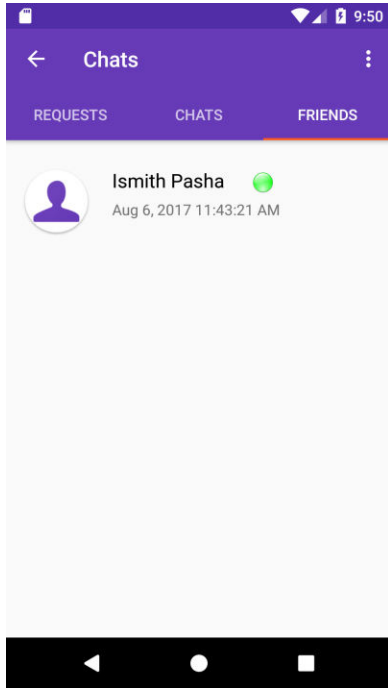


a)

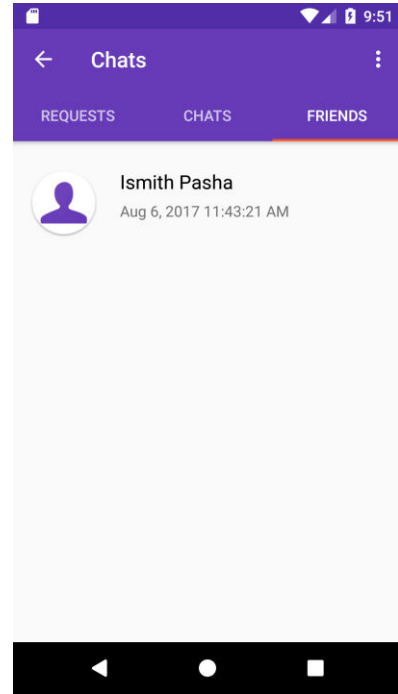


b)

Figure – 4.1: a) Full path of user with current location  
b) Only current location of user

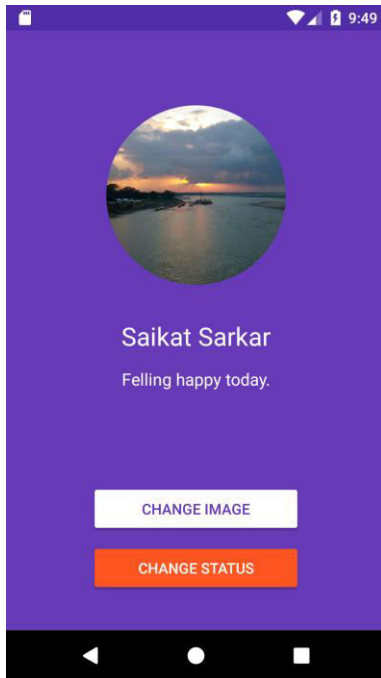


a)

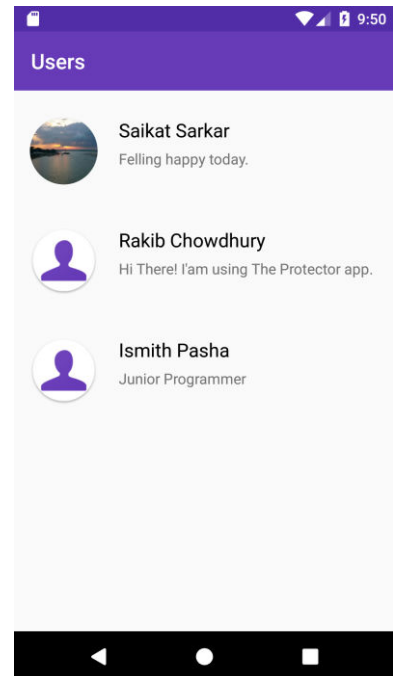


b)

Figure – 4.2: a) User Online, b) User Offline



a)



b)

Figure – 4.3: a) User Profile, b) User friend List

# **CHAPTER 5**

## **CONCLUSION**

### **5.1 OVERALL CONCLUSION**

In this project, we have developed energy and cost efficient android application framework that uses firebase cloud services as backend and share location at real time. We mainly tried to solve three problems, which are maintaining multiple backend server, excessive power drain and extra data load of in JSON parsing. To solve the multiple backend problem, we have proposed google provided firebase cloud platform for application backend. To solve the power drain problem, we proposed a method that decides location accuracy update option on the basis of battery power status. To solve to extra data parsing problem, we proposed a method to use single key with multiple concatenated values. We installed our application in multiple devices and it works as expected in all devices.

### **5.2 FUTURE WORKS**

Every project has scope for further improvement and this project is no exception. We have used firebase cloud functions to implement logics and conditions on data. As it is a new service and still is under development process, it has some limitations. Using further improvement of this platform geofencing, auto location alert, creating green zone, red zone features can be added. We have not used alternative location tracking APIs except google APIs, integrating other cloud services with this platform can be a better point to start improvement.

## REFERENCES

- [1] ENERGY-EFFICIENT LOCATION EASY TRACKING WITH ANDROID MOBILE PHONE BY USING GPS by Rajamoses.R and Sarooraj.R.B , Global institute for Research and Education, March-April, 2014.
- [2] Real Time Location Tracking Application based on Location Alarm by Adnaan Ghadiyali, Ankur Tiku, Sumeet Bandevar, Ruturaj Tengale , International Journal Of Engineering And Computer Science ISSN:2319-7242 Volume 4 Issue 4 April 2015, Page No. 11352-11355
- [3] Location tracking using Google Cloud Messaging on Android by Prof. C. J. Shelke<sup>1</sup>, Ms. Grishma R. Bhokare<sup>2</sup> , International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 12, December 2015
- [4] Android – A Cloud Computing for Vehicle Tracking System Using GPS by Mahalingam T. 1, Jeevitha.R2., Shunmuganathan K.L. in International Journal of Computer Applications in Engineering Sciences.
- [5] Real Time Tracking of Complete Transport System Using GPS, by Mr. Nilesh Manganakar Mr. Nikhil Pawar Mr. Prathamesh Pulaska in National Conference on New Horizons in IT - NCNHIT 2013.