GCM connect to Android Apps

Overview

Google Cloud Messaging (GCM) is one the service rendered by Google Play Services. GCM is an open service that allows the developers to send downstream messages (from servers to GCM enabled client apps) and upstream messages (from GCM enabled client apps to servers). The messages can either be light weight messages, informing the client app about a new message that needs to be fetched from the server. For instance, a New Email notification appears, informing that your app is out of sync with the back end. Or it could be a message containing up to 4kb of payload data, so that apps like instant messaging can access the message directly.

The GCM handles the following:

1. The entire aspects of queueing of messages.
2. Delivery of messages to and from the target client app.

Architectural Overview

A GCM implementation includes GCM connection servers, connection server, GCM enabled client app and a Database.

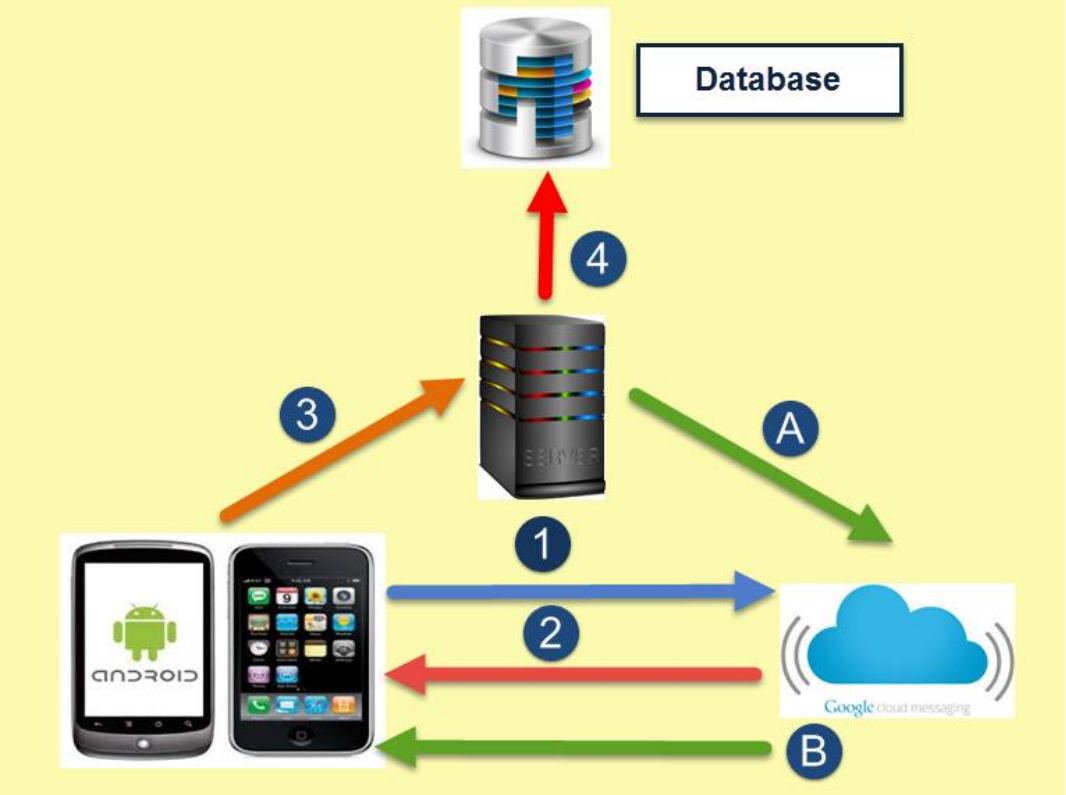


Figure 1: GCM Architecture

Components Interaction

1. The Android device sends Application ID to GCM server for registration.
2. On successful registration, GCM server issues Registration ID to Android device.
3. After receiving Registration ID, Android device sends Registration ID to server.
4. The server stores Registration ID in Database for future use.

A. While Push notification is necessary, the server sends a message to GCM server along with Android device Registration ID.

B. The GCM server delivers that message to respected mobile using device Registration ID.

Applications of GCM services

- GCM allows the applications to minimize data usage by eliminating the necessity for background polling.
- Display of notification to the user about the received information.

Enable GCM services

1. Open the browser. Type-In https://console.developers.google.com and click enter. The user is navigated to **Google Developers Console**.

   Note: The user should have to login to Gmail account.

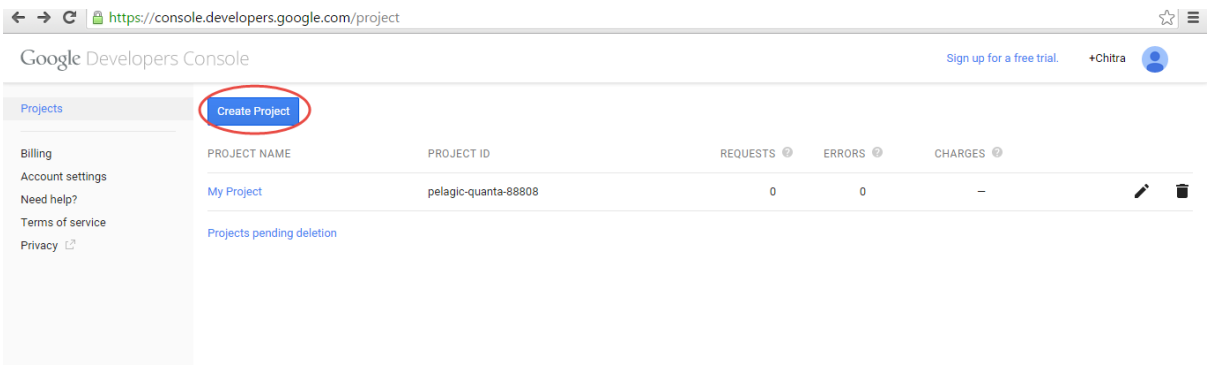


Figure 2: Google Developers Console

2. Click Create Project. The **New Project** window appears. Enter the name of the project in **Project Name** box and click **Create**.

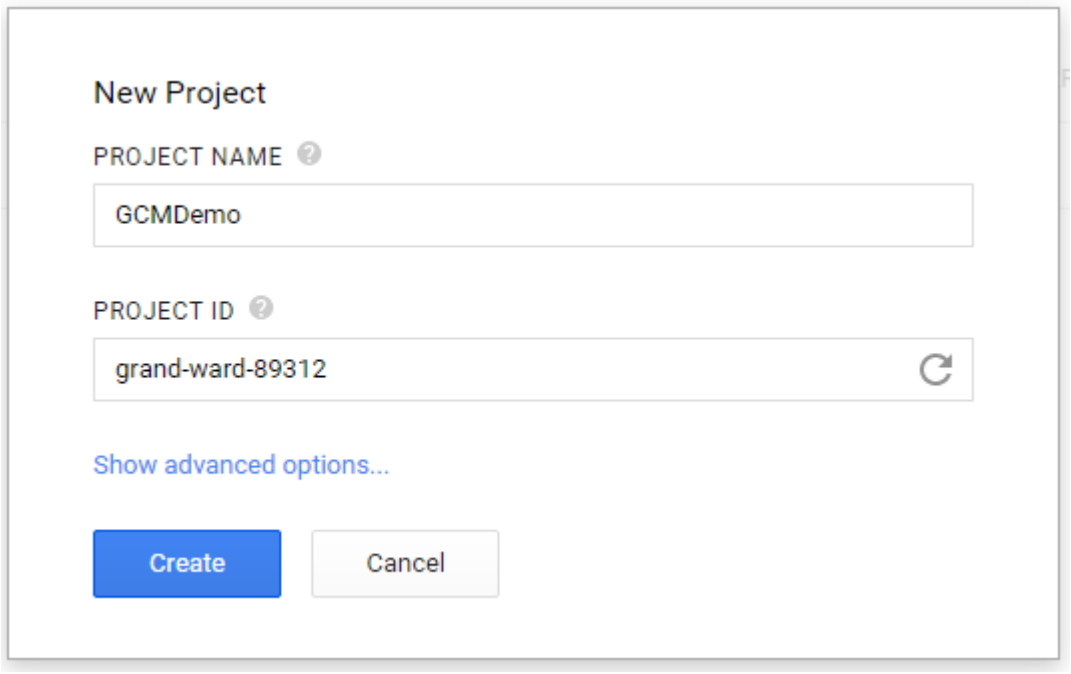


Figure 3: New Project dialog box

3. The created project appears in the list.

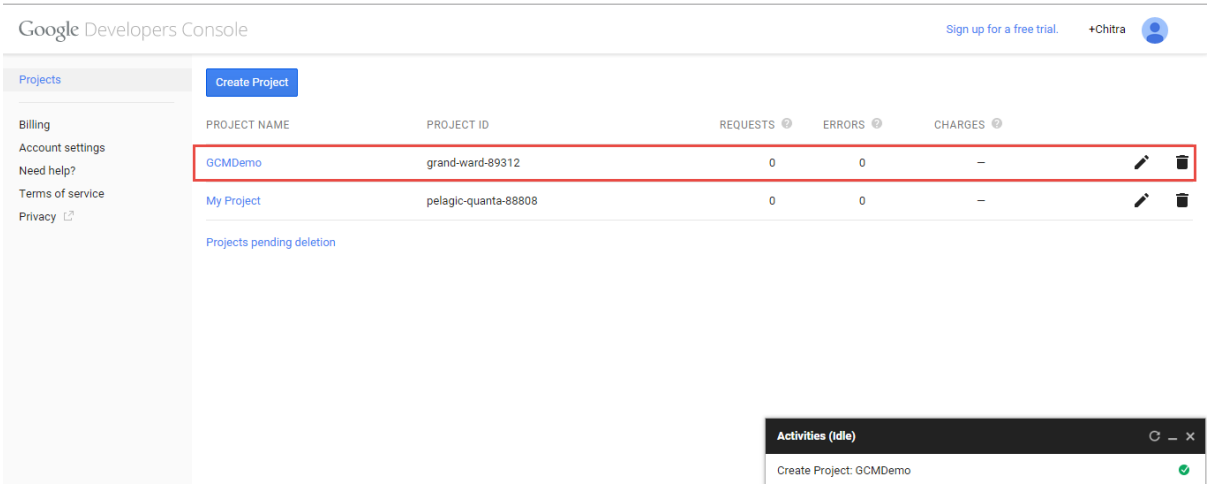


Figure 4: List containing the created project

4. Click the new project created. The Project Dashboard appears.

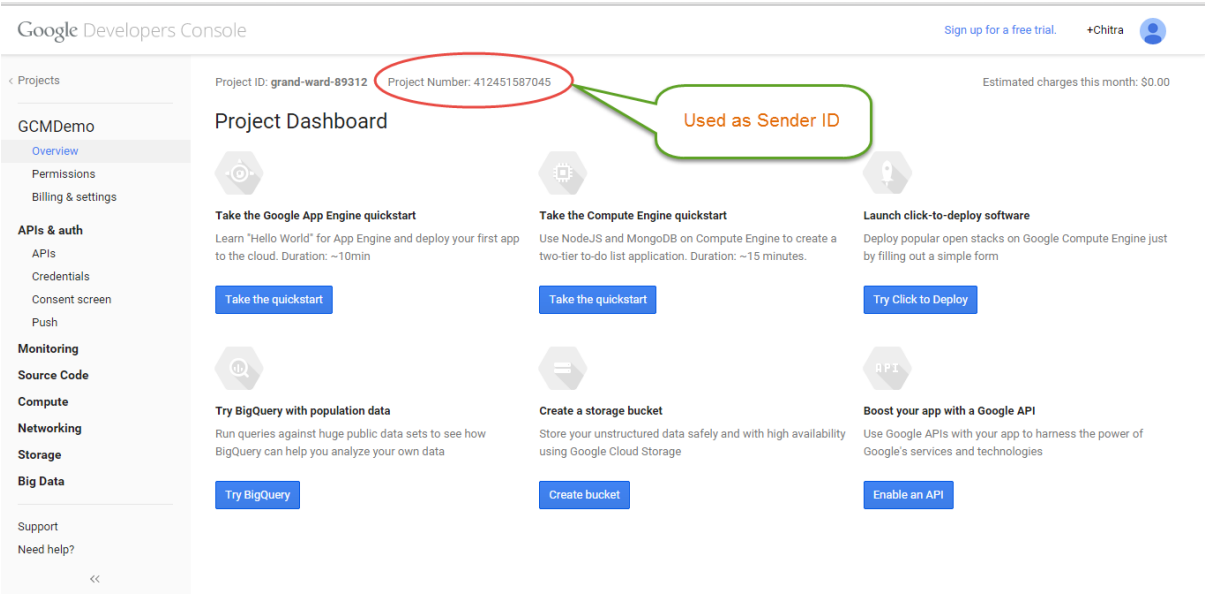


Figure 5: Project Dashboard

Note: The Project Number is used as Sender ID

5. On **APIs & auth** menu, Click **APIs**. Enabled APIs and Browse APIs list appears. Enable **Google Cloud messaging for android**.

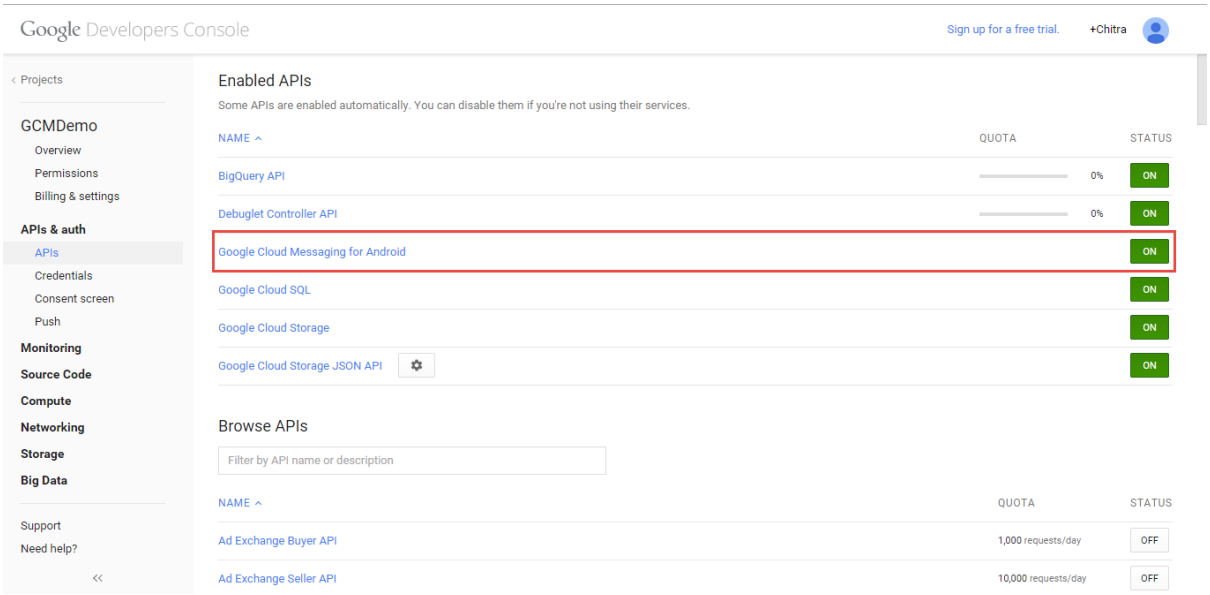


Figure 6: Enabling GCM for Android

Obtain an API key

1. On **APIs & auth** menu, click **Credentials**. The **OAuth** and **Public API** access section appears.

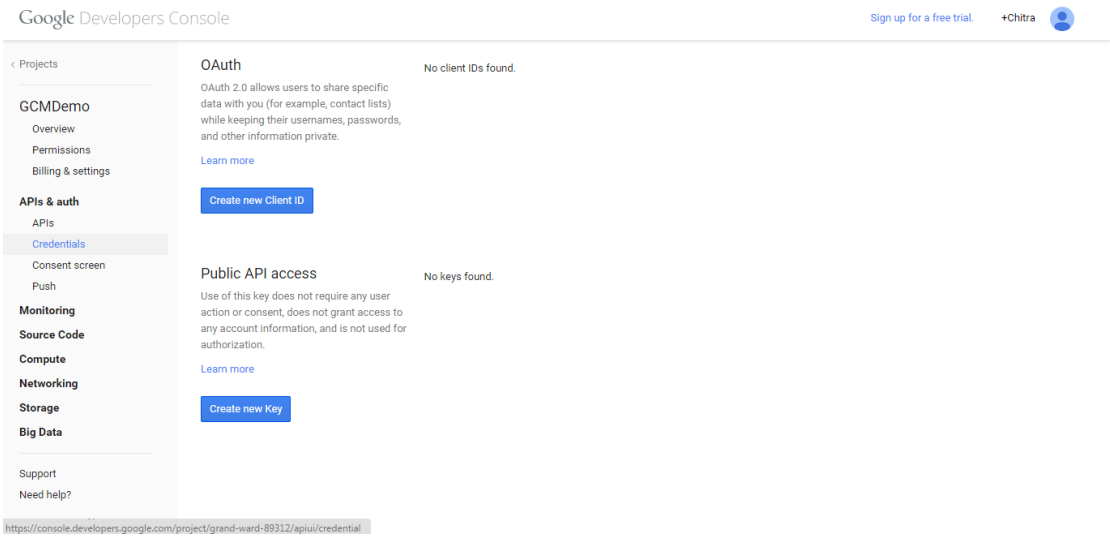


Figure 7: OAuth and Public API access section

2. Click **Create new key** on **Public API access** section. The **Create a new key** dialog box appears.

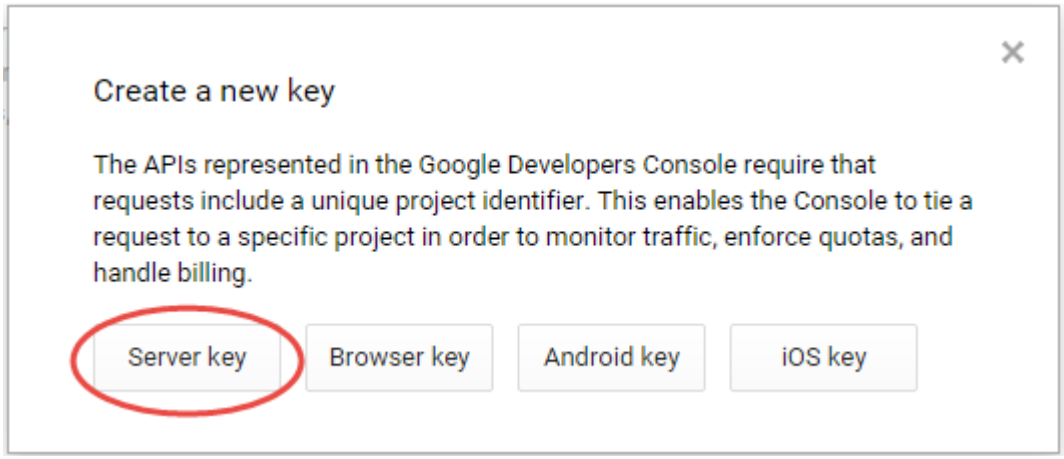


Figure 8: Create a new key dialog box

3. Click **Server key**, the **Create a server key and configure allowed IPs** dialog box appears. Enter the server IP address in **ACCEPT REQUESTS FROM THESE SERVER IP ADDRESSES** box. And click **Create**. The server key gets generated.
   Note: This API key is required to perform authentication in the app server.

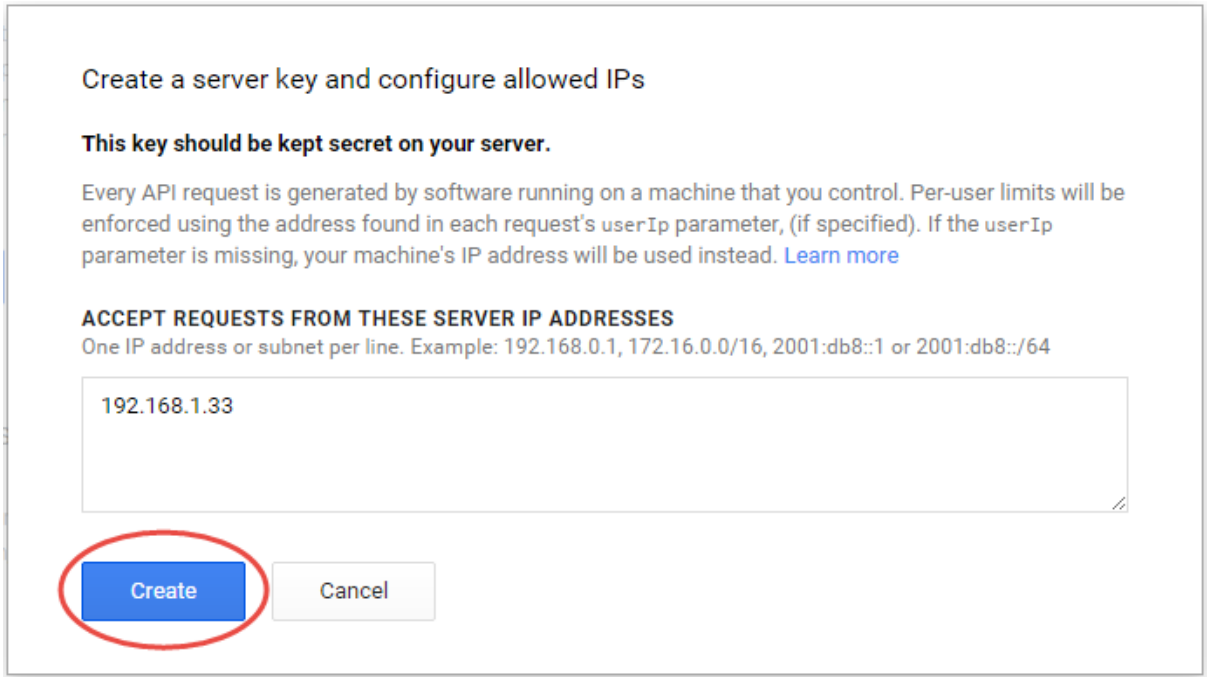


Figure 9: Configuration dialog box

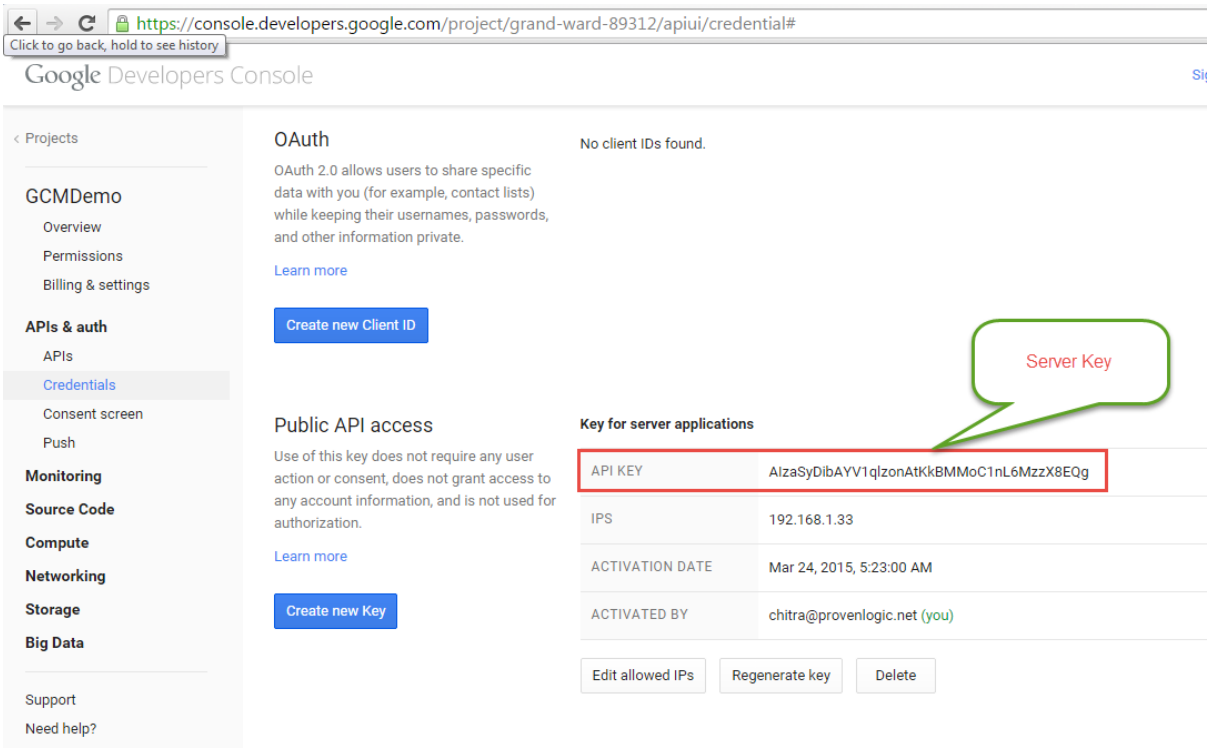


Figure 10: Server key generated

Way Forward

Once the above listed tasks are completed, the user is set to start to implement GCM. Implementation of GCM is performed in two steps:

1. Implementing GCM server.
2. Implementing GCM client.

Implementing GCM server

The server side of GCM includes two components

GCM connection servers – Accepts messages from 3rd party app server and sends them to a GCM enabled application that is the client app.

Prerequisite

- Server, preferably XAMPP localhost
- MYSQL database
- Enable curl

Perform the following steps to implement GCM server:

1. Install and run the XAMPP. Enable Apache and MySql.

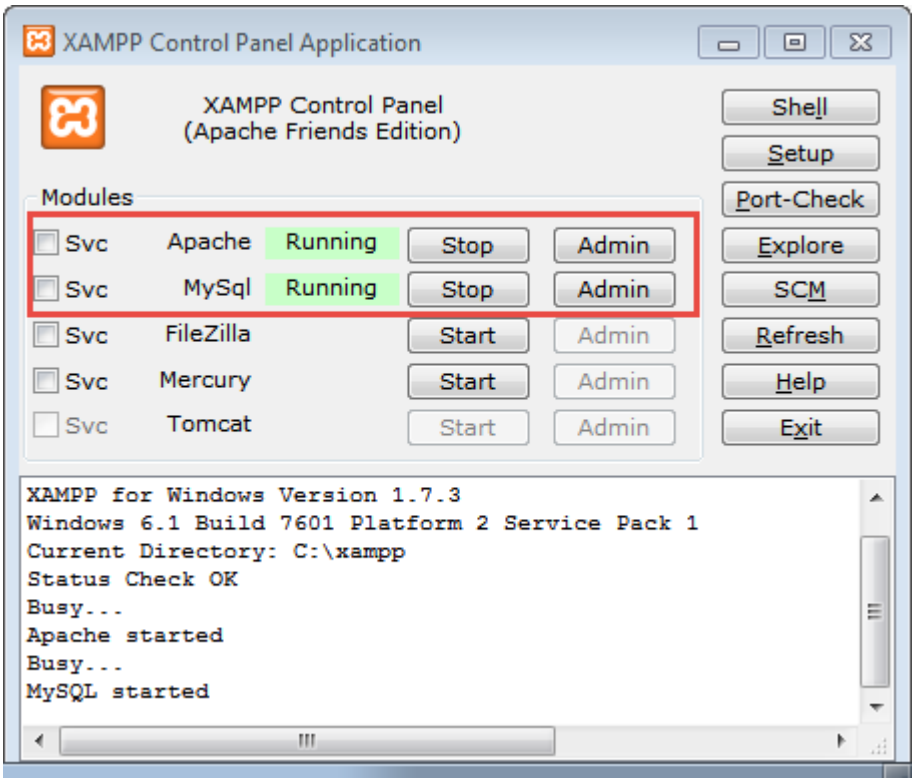


Figure 11: XAMPP Control Panel

2. Navigate to http://localhost/phpmyadmin/. Enter the name of the database in **Create new database** box and click **Create**.

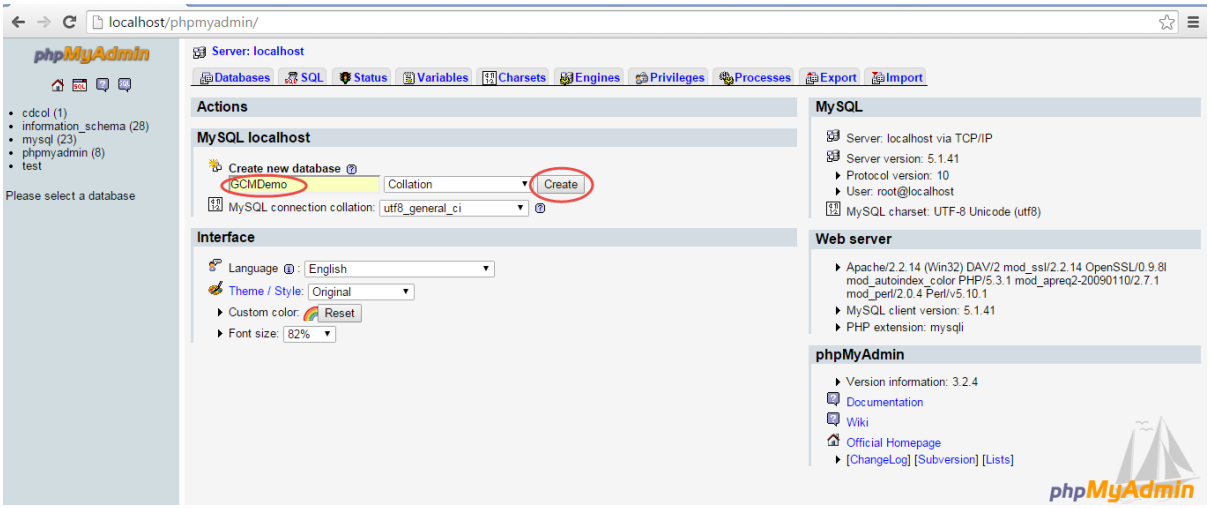


Figure 12: Creating a new Database

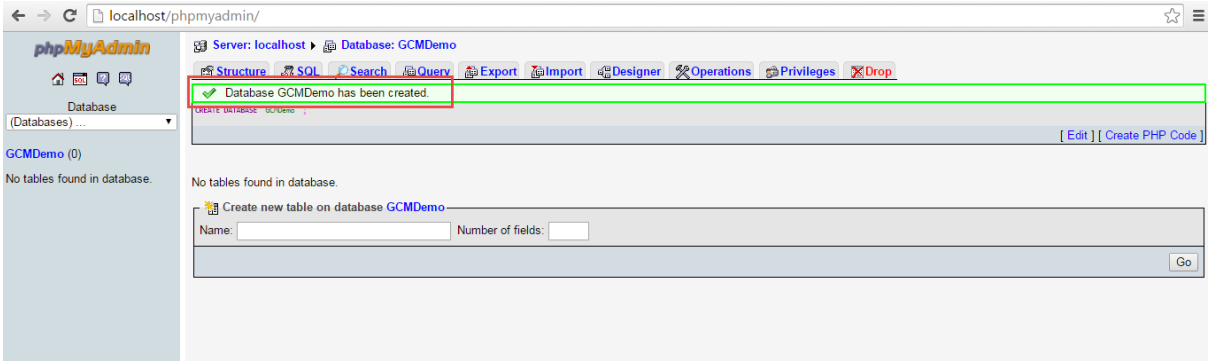


Figure 13: GCMDemo Database created

3. Import the **gcm_users.sql** file to the database created. i.e. GCMDemo database.

Query

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";

SET time_zone = "+00:00";

-- Database: `gcmdemo`

-- --------------------------------------------------------

-- Table structure for table `gcm_users`

--------------------------------------------------------


CREATE TABLE IF NOT EXISTS `gcm_users` (

  `id` int(11) NOT NULL AUTO_INCREMENT,

  `gcm_regid` text,

  `name` varchar(50) NOT NULL,

  `email` varchar(255) NOT NULL,
```

```
  `created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=11 ;
```

Description

A table **gcm_users** is created, if the table does not exist. The gcm_users table has the following fields:

1. id – type integer, size 11, cannot be null and auto incremented. And declared as Primary key.
2. Gcm_regid – type text .
3. name – type varchar, size 50 and cannot be null.
4. created_at – type timestamp, cannot be null and set to current timestamp

   Note: The fields specified for the table should not change.

4. Create and include the following files in **C:\xampp\htdocs\gcm_demo**

   A. Config.php

   Code

```
<?php
/**
  * Database config variables
*/
 define("DB_HOST", "localhost");
 define("DB_USER", "root");
 define("DB_PASSWORD", "");
define("DB_DATABASE", "GCMDemo");


 /*
  * Google API Key
*/
 // Place  your Google API Key
define("GOOGLE_API_KEY", "AIzaSyDibAYV1qlzonAtKkBMMoC1nL6MzzX8EQg");
?>
```

Description

The **config.php** file contains declarations related to database and **Google API key**. This file handles database connections, mainly opens and closes database connection.

B. GCM.php

Code

```
/**
 * Sending Push Notification
 */
public function send_notification($registatoin_ids, $message) {
    // include config
    include_once './config.php';

    // Set POST variables
    $url = 'https://android.googleapis.com/gcm/send';

    $fields = array(
        'registration_ids' => $registatoin_ids,
        'data' => $message,
    );

    $headers = array(
        'Authorization: key=' . GOOGLE_API_KEY,
        'Content-Type: application/json'
    );
    // Open connection
    $ch = curl_init();

    // Set the url, number of POST vars, POST data
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
```

```
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

        // Disabling SSL Certificate support temporarly
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));

        // Execute post
        $result = curl_exec($ch);
        if ($result === FALSE) {
            die('Curl failed: ' . curl_error($ch));
        }

        // Close connection
        curl_close($ch);
        echo $result;
    }
}
?>
```

Description

This file contains function to send push notification. The **$registatoin_ids** and **$message** is passed to **send_notification** function. In function **send_notification**, **config.php** is included once, POST variables are set such as https://android.googleapis.com/gcm/send is assigned to **$url**, an array containing registration ids and data is assigned to **$fields**, an array containing Authorization: key and Content-Type is assigned to **$headers**. Initialize a curl session by assigning **curl_init()** to **$ch**. Various options for the session are set. The SSL certificate support is temporally disabled. The **$ch** is passed to **curl_exec()**. The curl session is executed using **curl_exec($ch)**. The curl session is closed by **curl_close($ch)**.

C. db_functions.php

Code

```
class DB_Functions {
    private $db;

    //put your code here
```

```
    // constructor
    function __construct() {
        include_once './db_connect.php';
        // connecting to database
        $this->db = new DB_Connect();
        $this->db->connect();
    }
    // destructor
    function __destruct() {

    }

    /**
     * Storing new user
     * returns user details
     */
    public function storeUser($name, $email, $gcm_regid) {
        // insert user into database
        $result = mysql_query("INSERT INTO gcm_users(name, email, gcm_regid,
created_at) VALUES('$name', '$email', '$gcm_regid', NOW())");
        // check for successful store
        if ($result) {
            // get user details
            $id = mysql_insert_id(); // last inserted id
            $result = mysql_query("SELECT * FROM gcm_users WHERE id = $id") or
die(mysql_error());
            // return user details
            if (mysql_num_rows($result) > 0) {
                return mysql_fetch_array($result);
            } else {
                return false;
            }
```

```
        } else {
            return false;
        }
    }

    /**
     * Get user by email and password
     */
    public function getUserByEmail($email) {
        $result = mysql_query("SELECT * FROM gcm_users WHERE email = '$email' LIMIT
1");
        return $result;
    }

    /**
     * Getting all users
     */
    public function getAllUsers() {
        $result = mysql_query("select * FROM gcm_users");
        return $result;
    }

    /**
     * Check user is existed or not
     */
    public function isUserExisted($email) {
        $result = mysql_query("SELECT email from gcm_users WHERE email = '$email'");
        $no_of_rows = mysql_num_rows($result);
        if ($no_of_rows > 0) {
            // user existed
            return true;
```

```
        } else {
            // user not existed
            return false;
        }
    }
    public function getGcmID($email) {
        $result = mysql_query("SELECT * from gcm_users WHERE email = '$email'");
        $no_of_rows = mysql_num_rows($result);
        if ($no_of_rows > 0) {
            $user_details = mysql_fetch_assoc($result);
            return $user_details['gcm_regid'] ;
        } else {
            // user not existed
            return "";
            }
     }
}
?>
```

Description

This file contains functions to perform database operations such as Create, Read, Update and delete.

D. db_connect.php

Code

```
<?php

class DB_Connect {

    // constructor
    function __construct() {

    }
```

```
    // destructor
    function __destruct() {
        // $this->close();
    }

    // Connecting to database
    public function connect() {
        require_once 'config.php';
        // connecting to mysql
        $con = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
        // selecting database
        mysql_select_db(DB_DATABASE);

        // return database handler
        return $con;
    }

    // Closing database connection
    public function close() {
        mysql_close();
    }
}
?>
```

Description

This file handles database connections, mainly opens and closes connection.

E. register.php

Code

```
<?php

// response json
```

```
$json = array();

/**
 * Registering a user device
 * Store reg id in users table
 */
if (isset($_GET["name"]) && isset($_GET["email"]) && isset($_GET["regId"])) {
   $name = $_GET["name"];
   $email = $_GET["email"];
   $gcm_regid = $_GET["regId"]; // GCM Registration ID
   // Store user details in db
   include_once './db_functions.php';
   include_once './GCM.php';

   $db = new DB_Functions();
   $gcm = new GCM();

   $res = $db->storeUser($name, $email, $gcm_regid);

   if($res){
      echo "success";
   }else{
      echo "failure";
   }
   } else {
   // user details missing
   echo "missing";
}
?>
```

Description

The above mentioned code deals with receiving the requests from android device and storing the **reg_id** in **gcm_users** table. In the above mentioned code, the name, email and regId is checked. If all are set, then name is fetched by get method and assigned to **$name**, email is fetched by get method and assigned to **$email** and regId is fetched by get method and assigned to **$gcm_regid**. To store the user details in database, **db_functions.php** and **GCM.php** files are included once. An instance of **DB_Functions()** class is assigned to **$db**. An instance of **GCM()** class is assigned to **$gcm**. The **$name**, **$email** and **$gcm_regid** is passed to **storeUser** function in **DB_Functions()** class and assigned to **$res**. If the data is saved successfully, then success is displayed else failure is displayed. Else missing is displayed.

F. sendChatmessage.php

Code

```
<?php
if (isset($_GET["email_id"]) && isset($_GET["message"])) {
   $email = $_GET["email_id"];
   $message = $_GET["message"];
   include_once './GCM.php';
     include_once './db_functions.php';
     $gcm = new GCM();
     $db = new DB_Functions();
   $regId=$db->getGcmID($email);
   $registatoin_ids = array($regId);
   $message = array("text_message" => $message);
   $result = $gcm->send_notification($registatoin_ids, $message);


   echo $result;
}
?>
```

Description

The above mentioned code is used to send chat messages. In the above code, the email id and message is checked. If both them exist on input, then email id is fetched by get method and assigned to $email and message is fetched by get method and assigned to **$message**. The **GCM.php** and **db_functions.php** are included once. An instance of **GCM()** is assigned to **$gcm**. An instance of **DB_Functions()** is assigned to **$db**. The

$email is passed to **getGcmID** function of **DB_Functions** class and assigned to **$regId**. An array containing **$regId** is assigned to **$registatoin_ids**. An array containing text message is assigned to **$message**. The **$registatoin_ids** and **$message** is passed to **send_notification** function of GCM class and assigned to **$result**. And the **$result** is displayed.

Implementing GCM client

Perform the following steps to implement GCM client:

1. Launch the Android Studio.
2. On the File menu, click **New Project**. The **New Project** dialog box appears.

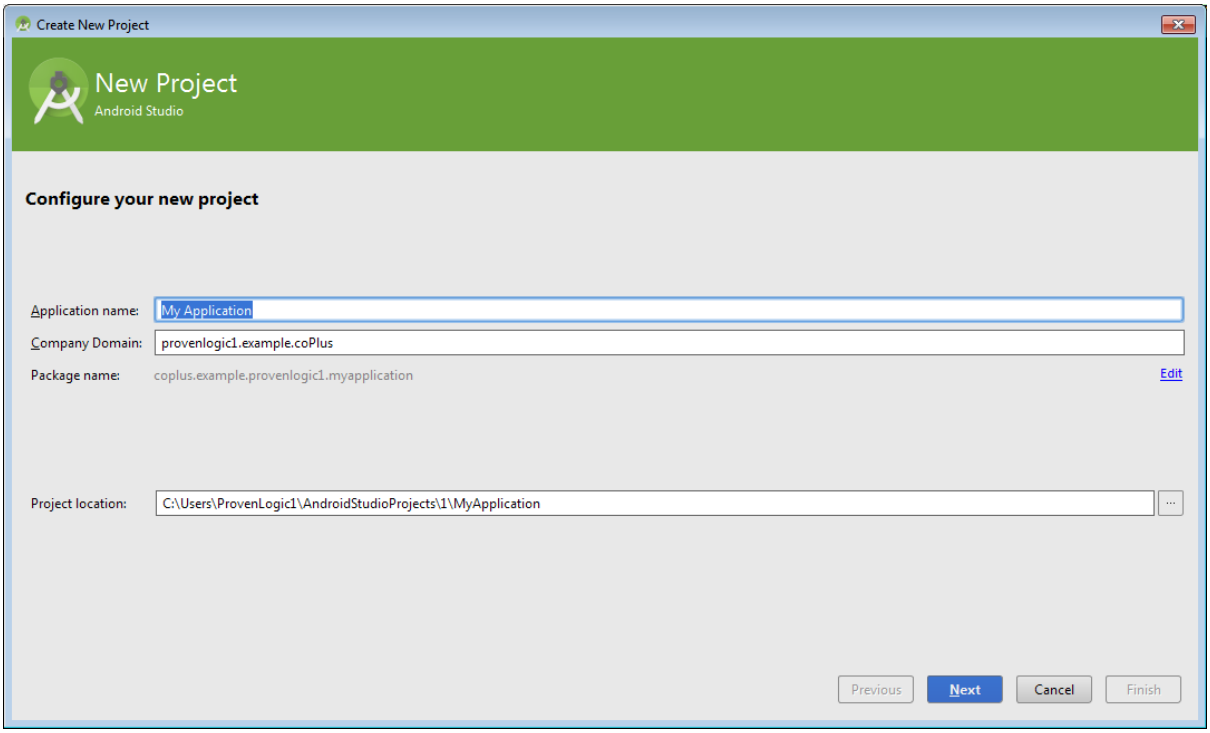


Figure 14: New Project dialog box

3. Enter the application name in **Application Name** box. Enter the company domain in **Company Domain** box. Choose the project location and click **Next**. The **Target Android Devices** dialog box appears.

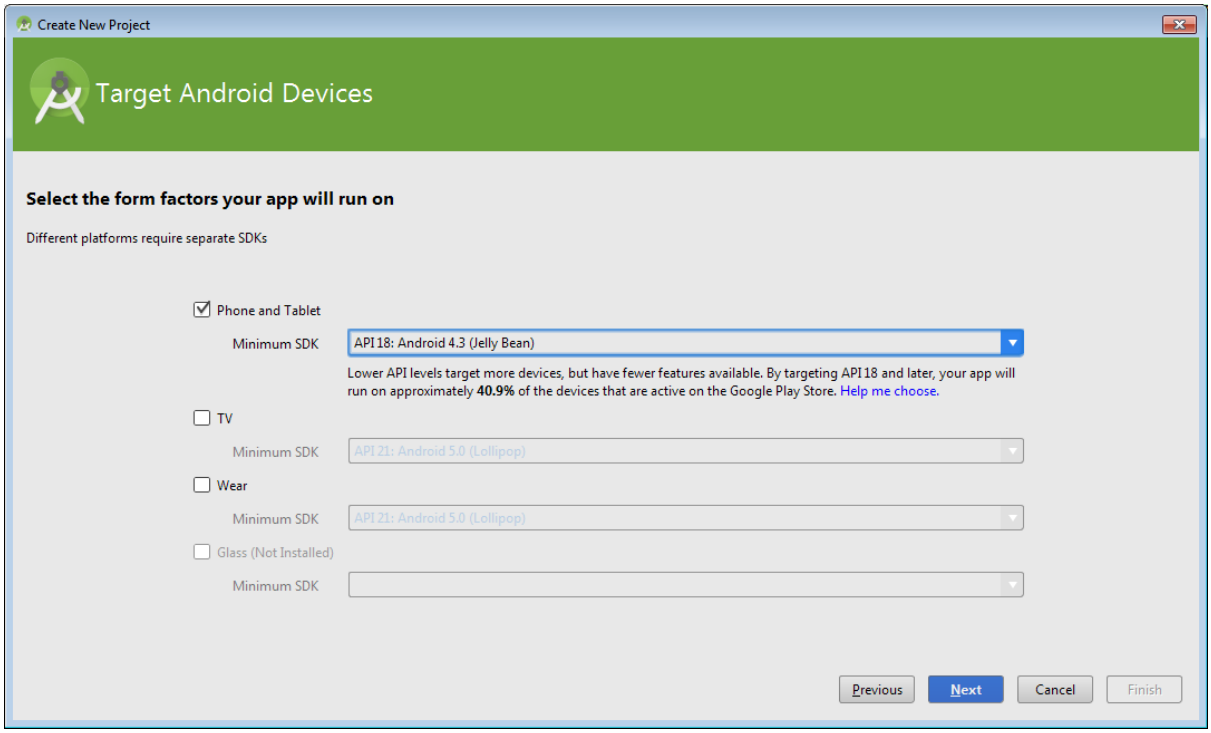


Figure 15: Target Android Devices dialog box

4. In the **Target Android Devices** dialog box, check **Phone and Tablet**. Select the minimum SDK from **Minimum SDK** list. Click Next. The **Add an activity to Mobile** dialog box appears.

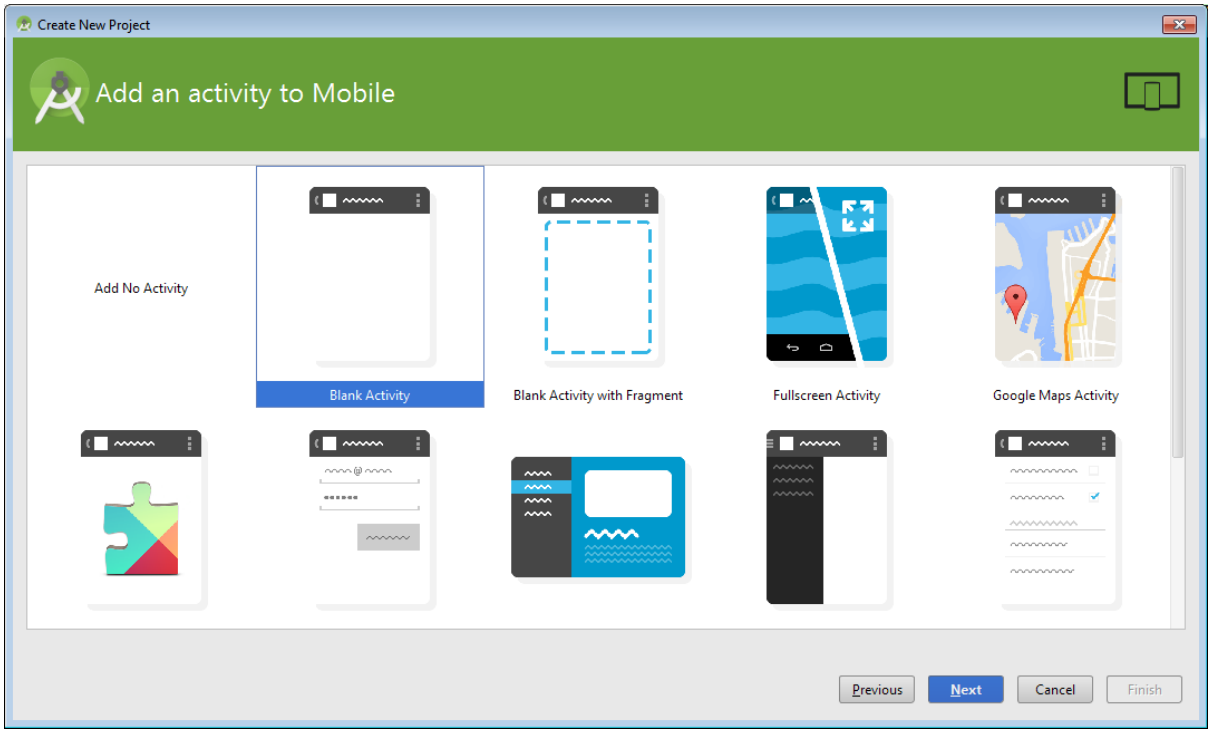


Figure 16: Add an activity to Mobile dialog box

5. In the **Add an activity to Mobile** dialog box, select the required activity and click **Next**. The **Customize the Activity** dialog box appears.

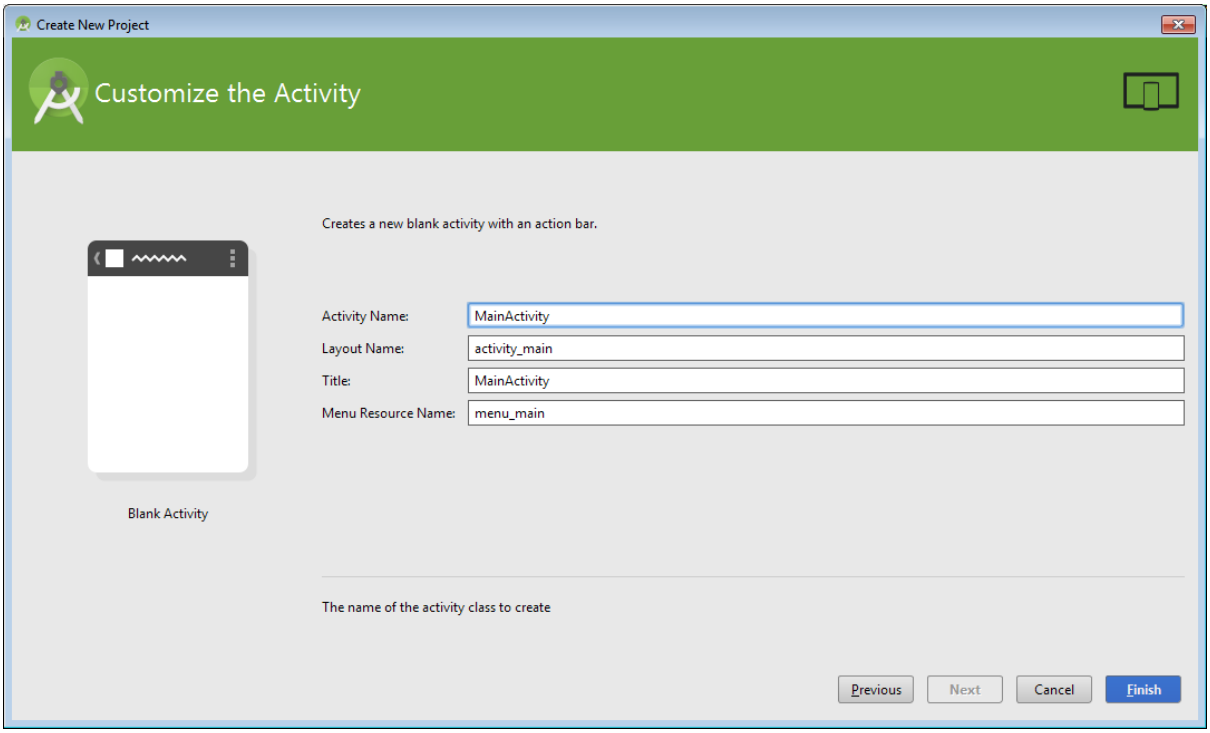


Figure 17: Customize the Activity dialog box

6. In the **Customize the Activity dialog** box, enter the activity name in **Activity Name** box. Enter the layout name in **Layout Name** box. Enter the title in **Title** box. Enter menu resource name in **Menu Resource Name** box. Click **Finish**. The new project **My Application** is created successfully.

7. Navigate to **Gradle Scripts** and open **build.gradle(Module: app)**. Add the following code under dependencies.

   Code

   ```
   compile 'com.google.android.gms:play-services:3.1.+'
   ```

   Description

   The above code is added to avail the service of Google Play services to your project.

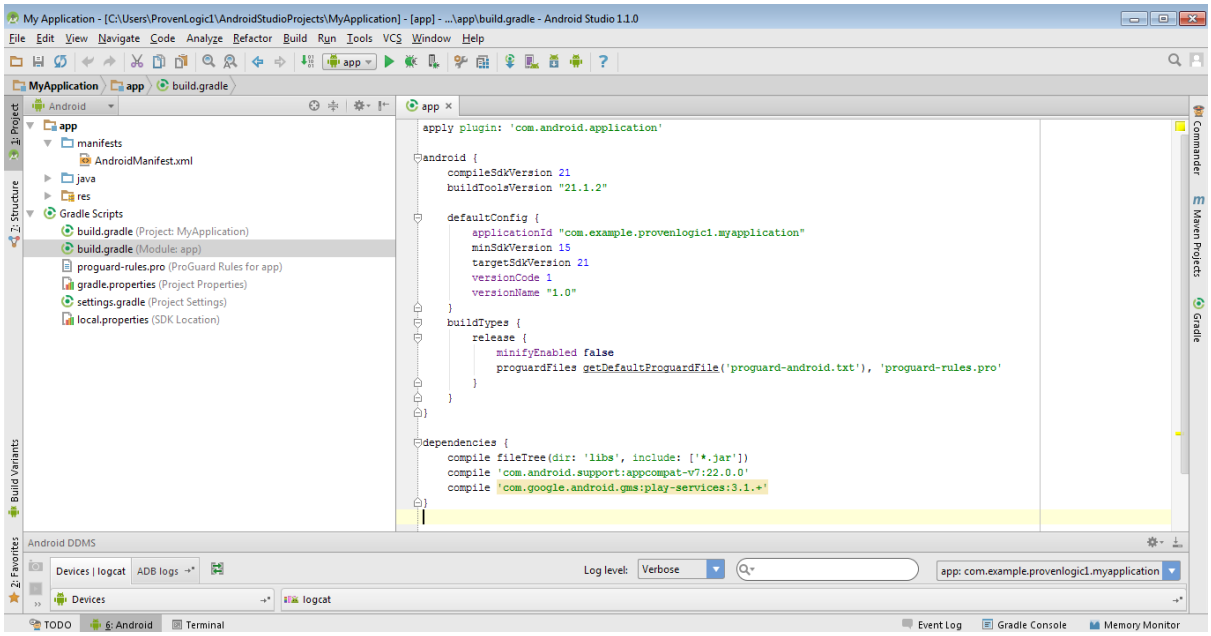


Figure 18:build.gradle(Module: app) file

8. Navigate to **app/manifests** and open **AndroidManifest.xml** and add the following code.

Code

```
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.GET_ACCOUNTS" />

<uses-permission android:name="android.permission.WAKE_LOCK" />

<uses-permission android:name="com.example.gcm.permission.C2D_MESSAGE" />

<application ...>


  <receiver
      android:name=".GcmBroadcastReceiver"
      android:permission="com.google.android.c2dm.permission.SEND" >


       <intent-filter>


         <action android:name="com.google.android.c2dm.intent.RECEIVE" />
          <category android:name="com.example.gcm" />


     </intent-filter>


     </receiver>
```

```
    <service android:name=".GcmIntentService" />

  </application>
```

Description

- The **com.google.android.c2dm.permission.RECEIVE** permission is included to allow the Android application to register and receive messages.
- The **android.permission.INTERNET** permission is included to allow the Android application to send the registration ID to the 3rd party server.
- The **android.permission.get_accounts** is required for registration to GCM.
- With **android.permission.WAKE_LOCK** permission the Android application can wake the processor on arrival of a new message.
- The **com.example.gcm.permission.C2D_MESSAGE** is included to prevent other Android applications from registering and receiving the Android application's messages. The permission must be specified exactly the way it is specified here else Android application will not receive the messages.
- The receiver must require **com.google.android.c2dm.permission.SEND** permission to allow only GCM framework to send message.
- The **WakefulBroadcastReceiver** permits the work of handling the GCM messages to IntentService. This service ensures that the device does not go back to sleep in the process.

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.provenlogic1.myapplication" >

    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />

    <permission
        android:name="com.example.gcm.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />

    <uses-permission android:name="com.example.gcm.permission.C2D_MESSAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver
            android:name=".GcmBroadcastReceiver"
            android:exported="true"
            android:permission="com.google.android.c2dm.permission.SEND" >
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.RECEIVE" />

                <category android:name="com.example.provenlogic1.myapplication" />
            </intent-filter>
        </receiver>

        <service android:name=".GcmIntentService" />

        <activity
            android:name=".ChatActivity"
            android:label="ChatActivity" >
        </activity>
    </application>

</manifest>
```

Figure 19: AndroidManifest.xml file

9. Navigate to **app\src\main\java\com.example.provenlogic1.myapplication** and add the following classes:

    - **GcmBroadcastReceiver**

      The **BroadcastReceiver** receives a device wakeup event and allocates the work to a service by ensuring that the device does not go back to sleep during the process.

Code

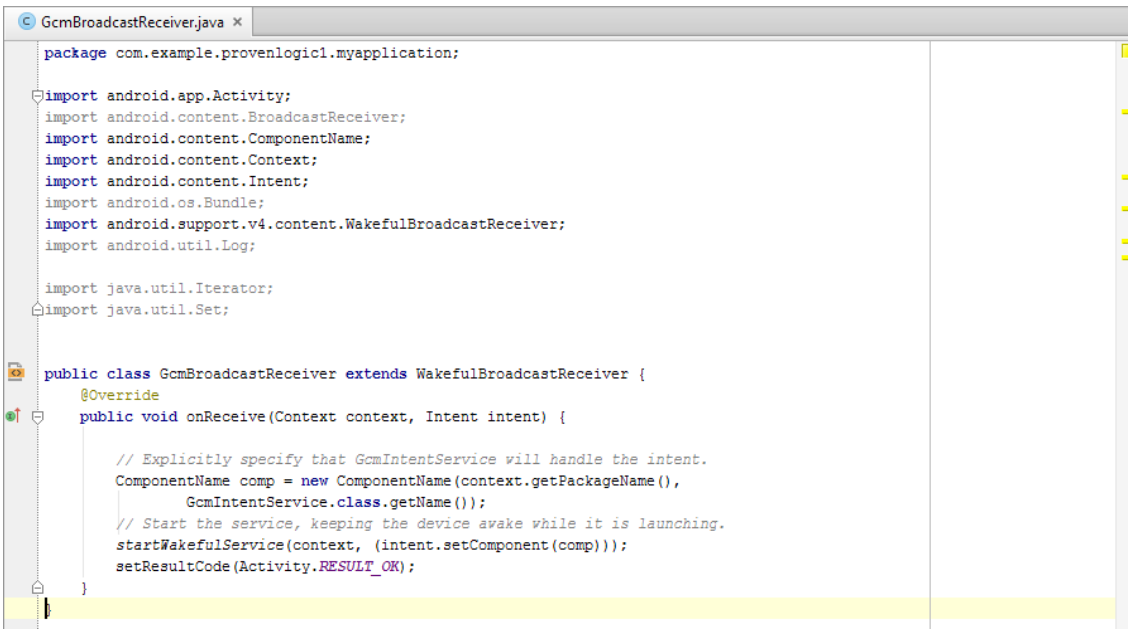
```
public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {


    // Explicitly specify that GcmIntentService will handle the intent.
    ComponentName comp = new ComponentName(context.getPackageName(),
          GcmIntentService.class.getName());
    // Start the service, keeping the device awake while it is launching.
    startWakefulService(context, (intent.setComponent(comp)));
    setResultCode(Activity.RESULT_OK);
  }
```

Descriprition

A **WakefulBroadcastReceiver** uses the method **startWakefulService()** to start the service that does the work. The intent that is passed with **startWakefulService()** holds an extra identifying the wake lock. The intent that is passes as a parameter is the same intent that the **WakefulBroadcastReceiver** originally had passed in.

In the above mentioned code, an instance of a class containing package name and intentService is assigned to comp object of **ComponentName** class. The **startWakefulService(context, (intent.setComponent(comp)))** Starts the service, keeping the device awake when it is launching



```
GcmBroadcastReceiver.java ×
package com.example.provenlogic1.myapplication;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.content.WakefulBroadcastReceiver;
import android.util.Log;

import java.util.Iterator;
import java.util.Set;


public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

        // Explicitly specify that GcmIntentService will handle the intent.
        ComponentName comp = new ComponentName(context.getPackageName(),
                GcmIntentService.class.getName());
        // Start the service, keeping the device awake while it is launching.
        startWakefulService(context, (intent.setComponent(comp)));
        setResultCode(Activity.RESULT_OK);
    }
}
```

Figure 20: GcmBroadcastReceiver.java file

- **GcmIntentServices**

  IntentService is a base class for services. This handles all the asynchronous requests on demand. The IntentService can perform the following:

  1. Receive the intents.
  2. Launch a worker thread.
  3. Stop the service when required.

  Code

```
public class GcmIntentService extends IntentService {
 public static final int NOTIFICATION_ID = 1;
 private NotificationManager mNotificationManager;
 NotificationCompat.Builder builder;
 String TAG="pavan";

 public GcmIntentService() {
 super("GcmIntentService");
 }

 @Override
 protected void onHandleIntent(Intent intent) {
    Bundle extras = intent.getExtras();
    GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
    // The getMessageType() intent parameter must be the intent you received
    // in your BroadcastReceiver.
    String messageType = gcm.getMessageType(intent);

    Log.d("pavan","in gcm intent message "+messageType);
    Log.d("pavan","in gcm intent message bundle "+extras);

    if (!extras.isEmpty()) {  // has effect of unparcelling Bundle
       /*
        * Filter messages based on message type. Since it is likely that GCM
```

```
         * will be extended in the future with new message types, just ignore
         * any message types you're not interested in, or that you don't
         * recognize.
         */
        if (GoogleCloudMessaging.
                MESSAGE_TYPE_SEND_ERROR.equals(messageType)) {
            sendNotification("Send error: " + extras.toString());
        } else if (GoogleCloudMessaging.
                MESSAGE_TYPE_DELETED.equals(messageType)) {
            sendNotification("Deleted messages on server: " +
                    extras.toString());
            // If it's a regular GCM message, do some work.
        } else if (GoogleCloudMessaging.
                MESSAGE_TYPE_MESSAGE.equals(messageType)) {

            String recieved_message=intent.getStringExtra("text_message");
            sendNotification("message recieved :" +recieved_message);

            Intent sendIntent =new Intent("message_recieved");
            sendIntent.putExtra("message",recieved_message);
            LocalBroadcastManager.getInstance(this).sendBroadcast(sendIntent);

        }
    }
    // Release the wake lock provided by the WakefulBroadcastReceiver.
    GcmBroadcastReceiver.completeWakefulIntent(intent);
}

// Put the message into a notification and post it.
// This is just one simple example of what you might choose to do with
// a GCM message.
```

```
private void sendNotification(String msg) {
    mNotificationManager = (NotificationManager)
            this.getSystemService(Context.NOTIFICATION_SERVICE);

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
            new Intent(this, MainActivity.class), 0);

    NotificationCompat.Builder mBuilder =
            new NotificationCompat.Builder(this)
                    .setSmallIcon(R.drawable.common_signin_btn_text_disabled_dark)
                    .setContentTitle("GCM Notification")
                    .setStyle(new NotificationCompat.BigTextStyle()
                            .bigText(msg))
                    .setContentText(msg);

    mBuilder.setContentIntent(contentIntent);
    mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
}
```

Description

This class will handle the broadcast, create a notification and launch the chat activity.

```java
package com.example.provenlogic1.myapplication;

import android.app.IntentService;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.SystemClock;
import android.support.v4.app.NotificationCompat;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import com.google.android.gms.gcm.GoogleCloudMessaging;

public class GcmIntentService extends IntentService {
    public static final int NOTIFICATION_ID = 1;
    private NotificationManager mNotificationManager;
    NotificationCompat.Builder builder;
    String TAG="pavan";

    public GcmIntentService() {
        super("GcmIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Bundle extras = intent.getExtras();
        GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
        // The getMessageType() intent parameter must be the intent you received
        // in your BroadcastReceiver.
        String messageType = gcm.getMessageType(intent);

        Log.d("pavan","in gcm intent message "+messageType);
        Log.d("pavan","in gcm intent message bundle "+extras);

        if (!extras.isEmpty()) {  // has effect of unparcelling Bundle
            /*
             * Filter messages based on message type. Since it is likely that GCM
             * will be extended in the future with new message types, just ignore
             * any message types you're not interested in, or that you don't
             * recognize.
             */
            if (GoogleCloudMessaging.
                    MESSAGE_TYPE_SEND_ERROR.equals(messageType)) {
                sendNotification("Send error: " + extras.toString());
            } else if (GoogleCloudMessaging.
                    MESSAGE_TYPE_DELETED.equals(messageType)) {
                sendNotification("Deleted messages on server: " +
                        extras.toString());
                // If it's a regular GCM message, do some work.
            } else if (GoogleCloudMessaging.
                    MESSAGE_TYPE_MESSAGE.equals(messageType)) {

                String recieved_message=intent.getStringExtra("text_message");
                sendNotification("message recieved :" +recieved_message);

                Intent sendIntent =new Intent("message_recieved");
                sendIntent.putExtra("message",recieved_message);
                LocalBroadcastManager.getInstance(this).sendBroadcast(sendIntent);

            }
        }
        // Release the wake lock provided by the WakefulBroadcastReceiver.
        GcmBroadcastReceiver.completeWakefulIntent(intent);
    }

    // Put the message into a notification and post it.
    // This is just one simple example of what you might choose to do with
    // a GCM message.
    private void sendNotification(String msg) {
        mNotificationManager = (NotificationManager)
                this.getSystemService(Context.NOTIFICATION_SERVICE);

        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, MainActivity.class), 0);

        NotificationCompat.Builder mBuilder =
                new NotificationCompat.Builder(this)
                        .setSmallIcon(R.drawable.common_signin_btn_text_disabled_dark)
                        .setContentTitle("GCM Notification")
                        .setStyle(new NotificationCompat.BigTextStyle()
                                .bigText(msg))
                        .setContentText(msg);

        mBuilder.setContentIntent(contentIntent);
        mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
    }
}
```

Figure 21: GcmIntentService.java file

- **MainActivity**

  The main activity code is a java file. This is responsible for the view that comes up when you first open the app.

  Code

```
public class MainActivity extends ActionBarActivity {
EditText editText_user_name;
EditText editText_email;
Button button_login;
static final String TAG = "pavan";
TextView mDisplay;
GoogleCloudMessaging gcm;
AtomicInteger msgId = new AtomicInteger();
SharedPreferences prefs;
Context context;
String regid;
String msg;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  context = getApplicationContext();
  if(isUserRegistered(context)){
    startActivity(new Intent(MainActivity.this,ChatActivity.class));
    finish();

  }else {

    editText_user_name = (EditText) findViewById(R.id.editText_user_name);
    editText_email = (EditText) findViewById(R.id.editText_email);
```

```
        button_login = (Button) findViewById(R.id.button_login);
        button_login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendRegistrationIdToBackend();


            }
        });
        // Check device for Play Services APK. If check succeeds, proceed with
        //  GCM registration.

        if (checkPlayServices()) {
            gcm = GoogleCloudMessaging.getInstance(this);
            regid = getRegistrationId(context);

            if (regid.isEmpty()) {
                registerInBackground();
            }
        } else {
            Log.i("pavan", "No valid Google Play Services APK found.");
        }
    }
}

private boolean checkPlayServices() {
    int resultCode = GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);
    if (resultCode != ConnectionResult.SUCCESS) {
        if (GooglePlayServicesUtil.isUserRecoverableError(resultCode)) {
            GooglePlayServicesUtil.getErrorDialog(resultCode, this,
                    Util.PLAY_SERVICES_RESOLUTION_REQUEST).show();
        } else {
```

```
            Log.i(TAG, "This device is not supported.");
            finish();
        }
        return false;
    }
    return true;
}

private String getRegistrationId(Context context) {
    final SharedPreferences prefs = getGCMPreferences(context);
    String registrationId = prefs.getString(Util.PROPERTY_REG_ID, "");
    if (registrationId.isEmpty()) {
        Log.i(TAG, "Registration not found.");
        return "";
    }
    // Check if app was updated; if so, it must clear the registration ID
    // since the existing registration ID is not guaranteed to work with
    // the new app version.
    int registeredVersion = prefs.getInt(Util.PROPERTY_APP_VERSION,
Integer.MIN_VALUE);
    int currentVersion = getAppVersion(context);
    if (registeredVersion != currentVersion) {
        Log.i(TAG, "App version changed.");
        return "";
    }
    return registrationId;
}
private boolean isUserRegistered(Context context) {
    final SharedPreferences prefs = getGCMPreferences(context);
    String User_name = prefs.getString(Util.USER_NAME, "");
    if (User_name.isEmpty()) {
```

```
        Log.i(TAG, "Registration not found.");
        return false;
    }

    return true;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
/**
 * Substitute you own sender ID here. This is the project number you got
 * from the API Console, as described in "Getting Started."
 */
```

```
private void registerInBackground() {
new AsyncTask() {
    @Override
    protected String doInBackground(Object[] params) {
      try {

        if (gcm == null) {
          gcm = GoogleCloudMessaging.getInstance(MainActivity.this);
        }
        regid = gcm.register(Util.SENDER_ID);
        msg = "Device registered, registration ID=" + regid;

        // You should send the registration ID to your server over HTTP,
        //GoogleCloudMessaging gcm;/ so it can use GCM/HTTP or CCS to send
messages to your app.
        // The request to your server should be authenticated if your app
        // is using accounts.
       // sendRegistrationIdToBackend();

        // For this demo: we don't need to send it because the device
        // will send upstream messages to a server that echo back the
        // message using the 'from' address in the message.

        // Persist the registration ID - no need to register again.
        storeRegistrationId(context, regid);
      } catch (IOException ex) {
        msg = "Error :" + ex.getMessage();
        // If there is an error, don't just keep trying to register.
        // Require the user to click a button again, or perform
        // exponential back-off.
      }
```

```
            return msg;
        }
    }.execute();

}
private static int getAppVersion(Context context) {
    try {
        PackageInfo packageInfo = context.getPackageManager()
                .getPackageInfo(context.getPackageName(), 0);
        return packageInfo.versionCode;
    } catch (PackageManager.NameNotFoundException e) {
        // should never happen
        throw new RuntimeException("Could not get package name: " + e);
    }
}
private void storeRegistrationId(Context context, String regId) {
    final SharedPreferences prefs = getGCMPreferences(context);
    int appVersion = getAppVersion(context);
    Log.i(TAG, "Saving regId on app version " + appVersion);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString(Util.PROPERTY_REG_ID, regId);
    editor.putInt(Util.PROPERTY_APP_VERSION, appVersion);
    editor.commit();
}

private void storeUserDetails(Context context) {
    final SharedPreferences prefs = getGCMPreferences(context);
    int appVersion = getAppVersion(context);
    Log.i(TAG, "Saving regId on app version " + appVersion);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString(Util.EMAIL, editText_email.getText().toString());
```

```
        editor.putString(Util.USER_NAME, editText_user_name.getText().toString());
        editor.commit();
    }

    private SharedPreferences getGCMPreferences(Context context) {
        // This sample app persists the registration ID in shared preferences, but
        // how you store the registration ID in your app is up to you.
        return getSharedPreferences(MainActivity.class.getSimpleName(),
                Context.MODE_PRIVATE);
    }

   //  private RequestQueue mRequestQueue;
    private void sendRegistrationIdToBackend() {
        // Your implementation here.

        new SendGcmToServer().execute();

  // Access the RequestQueue through your singleton class.
        // AppController.getInstance().addToRequestQueue(jsObjRequest,
  "jsonRequest");
}
    private class SendGcmToServer extends AsyncTask<String, Void, String> {
        @Override
        protected void onPreExecute() {
            // TODO Auto-generated method stub
            super.onPreExecute();
        }

        @Override
        protected String doInBackground(String... params) {
            // TODO Auto-generated method stub
```

```
 String url =
Util.register_url+"?name="+editText_user_name.getText().toString()+"&email="+edi
tText_email.getText().toString()+"&regId="+regid;
        Log.i("pavan", "url" + url);
        OkHttpClient client_for_getMyFriends = new OkHttpClient();;
        String response = null;
        // String response=Utility.callhttpRequest(url);
        try {
            url = url.replace(" ", "%20");
            response = callOkHttpRequest(new URL(url),
                    client_for_getMyFriends);
            for (String subString : response.split("<script", 2)) {
                response = subString;
                break;
            }
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return response;
    }

    @Override
    protected void onPostExecute(String result) {
        // TODO Auto-generated method stub
        super.onPostExecute(result);
        //Toast.makeText(context,"response "+result,Toast.LENGTH_LONG).show();

        if (result != null) {
```

```
                if (result.equals("success")) {
                    storeUserDetails(context);
                    startActivity(new Intent(MainActivity.this, ChatActivity.class));
                    finish();

                } else {
            Toast.makeText(context, "Try Again" + result, Toast.LENGTH_LONG).show();
                }
            }else{
        Toast.makeText(context, "Check net connection ", Toast.LENGTH_LONG).show();
            }
        }
    }
        // Http request using OkHttpClient
        String callOkHttpRequest(URL url, OkHttpClient tempClient)
            throws IOException {

            HttpURLConnection connection = tempClient.open(url);
            connection.setConnectTimeout(40000);
            InputStream in = null;
            try {
                // Read the response.
                in = connection.getInputStream();
                byte[] response = readFully(in);
                return new String(response, "UTF-8");
            } finally {
                if (in != null)
                    in.close();
            }
        }
```

```
    byte[] readFully(InputStream in) throws IOException {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    byte[] buffer = new byte[1024];
    for (int count; (count = in.read(buffer)) != -1;) {
      out.write(buffer, 0, count);
    }
    return out.toByteArray();
  }
```

Description

If the user runs the application, UI screen appears. The user needs to specify the username, email and click login. Once the user clicks login, **onCreate** method is executed. In **onCreate** method, the user is checked for registration. If the user has registered, intentservice starts chat activity. Else the username is fetched from the editText_user_name textbox and assigned to editText_user_name object, email is fetched from editText_email textbox and assigned to editText_email object and button_id is fetched and assigned to button_login object. Once the user clicks the login button, **sendRegistrationIdToBackend**() function is executed. In **sendRegistrationIdToBackend()** function, **SendGcmToServer()** function is executed. The GCM RegId is fetched from **Util.java** if already created else if this is first run or a newer version of the application, a new ID is created. The application gives an error and Chat Activity is inaccessible if Google Play Services is not available on the device or there is some network issues. In the case of successful GCMID creation, a URL string is constructed using the username, email and GCM ID. And this URL string is passed to the server through HTTP and response is checked. If the response is successful, then context is passed to **storeUserDetails**()function and **storeUserDetails()** function is executed. In **storeUserDetails()** function, the details of the user is fetched using **getGCMPreferences(context)** and assigned to **SharedPreferences**. Using **prefs.edit()** function, **SharedPreferences** is edited. The email is accessed by **Util.EMAIL**, **editText_email.getText().toString()** and passed to **editor.putString**. The user name is accessed by **Util.USER_NAME**, **editText_user_name.getText().toString()** and passed to **editor.putString** and finally **SharedPreferences** is saved. Now the IntentService opens the chat activity.

- **ChatActivity**

It is like a placeholder for chat bubbles to appear. And it contains a list view.

Code

```
public class ChatActivity extends Activity {
  EditText editText_mail_id;
  EditText editText_chat_message;
  ListView listView_chat_messages;
```

```
    Button button_send_chat;
    List<ChatObject> chat_list;
    BroadcastReceiver recieve_chat;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chat);
        editText_mail_id= (EditText) findViewById(R.id.editText_mail_id);
        editText_chat_message= (EditText) findViewById(R.id.editText_chat_message);
        listView_chat_messages= (ListView)
findViewById(R.id.listView_chat_messages);
        button_send_chat= (Button) findViewById(R.id.button_send_chat);
        button_send_chat.setOnClickListener(new View.OnClickListener() {
            @Override
             public void onClick(View v) {
            // send chat message to server
                String message=editText_chat_message.getText().toString();
                showChat("sent",message);
                new SendMessage().execute();
                editText_chat_message.setText("");
            }
        });

        recieve_chat=new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
             String message=intent.getStringExtra("message");
               Log.d("pavan","in local braod "+message);
               showChat("recieve",message);
            }
        };
```

```
 LocalBroadcastManager.getInstance(this).registerReceiver(recieve_chat, new
IntentFilter("message_recieved"));

   }

   private void showChat(String type, String message){

      if(chat_list==null || chat_list.size()==0){

         chat_list= new ArrayList<ChatObject>();

      }

      chat_list.add(new ChatObject(message,type));

      ChatAdabter chatAdabter=new
ChatAdabter(ChatActivity.this,R.layout.chat_view,chat_list);

      listView_chat_messages.setAdapter(chatAdabter);

      //chatAdabter.notifyDataSetChanged();


   }

  @Override

   protected void onDestroy() {

      super.onDestroy();

   }

   private class SendMessage extends AsyncTask<String, Void, String> {

      @Override

      protected void onPreExecute() {

         // TODO Auto-generated method stub

         super.onPreExecute();

      }

      @Override

      protected String doInBackground(String... params) {

         // TODO Auto-generated method stub

         String url =
Util.send_chat_url+"?email_id="+editText_mail_id.getText().toString()+"&message=
"+editText_chat_message.getText().toString();

         Log.i("pavan", "url" + url);

         OkHttpClient client_for_getMyFriends = new OkHttpClient();;
```

```
        String response = null;
        // String response=Utility.callhttpRequest(url);

        try {
            url = url.replace(" ", "%20");
            response = callOkHttpRequest(new URL(url),
                    client_for_getMyFriends);
            for (String subString : response.split("<script", 2)) {
                response = subString;
                break;
            }
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return response;
    }
    @Override
    protected void onPostExecute(String result) {
        // TODO Auto-generated method stub
        super.onPostExecute(result);
        //Toast.makeText(context,"response "+result,Toast.LENGTH_LONG).show();
    }
}
// Http request using OkHttpClient
String callOkHttpRequest(URL url, OkHttpClient tempClient)
        throws IOException {
    HttpURLConnection connection = tempClient.open(url);
```

```
    connection.setConnectTimeout(40000);
    InputStream in = null;
    try {
        // Read the response.
        in = connection.getInputStream();
        byte[] response = readFully(in);
        return new String(response, "UTF-8");
    } finally {
        if (in != null)
            in.close();
    }
}

byte[] readFully(InputStream in) throws IOException {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    byte[] buffer = new byte[1024];
    for (int count; (count = in.read(buffer)) != -1;) {
        out.write(buffer, 0, count);
    }
    return out.toByteArray();
}
```

- **ChatObject**

  The chatobject contains all the chats pertaining to individual user.

  Code

```
public class ChatObject {
String message;
public String getType() {
 return type;
 }
 String type;
 public ChatObject(String message,String type) {
```

```
    this.message = message;
     this.type   = type;
    }

  public String getMessage() {
  return message;
    }

    public void setMessage(String message) {
    this.message = message;
    }
```

- **ChatAdabter**

  The chatAdapter populates the chat activity with individual chat messages that are stored in the chat object.

  Code

```
public class ChatAdabter extends ArrayAdapter<ChatObject> {
    List<ChatObject> chat_data;
    Context context;
    int resource;
    public ChatAdabter(Context context, int resource,List<ChatObject> chat_data) {
        super(context, resource, chat_data);
        this.chat_data=chat_data;
        this.context=context;
        this.resource=resource;
    }
       private class ViewHolder{
        TextView textView_left_chat;
        TextView textView_right_chat;
  }
```

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
ViewHolder holder = null;
  if(convertView==null){
    LayoutInflater inflater = LayoutInflater.from(context);
     convertView = inflater.inflate(R.layout.chat_view,null);
     holder = new ViewHolder();
     holder.textView_left_chat = (TextView)
convertView.findViewById(R.id.textView_left_chat);
     holder.textView_right_chat = (TextView)
convertView.findViewById(R.id.textView_right_chat);
     convertView.setTag(holder);
  }else{
     holder = (ViewHolder) convertView.getTag();
  }
  Log.d("pavan","type "+chat_data.get(position).getType());
  Log.d("pavan","message "+chat_data.get(position).getMessage());
  if(chat_data.get(position).getType().equals("sent")){
    holder.textView_left_chat.setText(chat_data.get(position).getMessage());
    holder.textView_right_chat.setVisibility(View.GONE);
    holder.textView_left_chat.setVisibility(View.VISIBLE);
     }else{
        holder.textView_right_chat.setText(chat_data.get(position).getMessage());
        holder.textView_left_chat.setVisibility(View.GONE);
        holder.textView_right_chat.setVisibility(View.VISIBLE);
     }
  return convertView;
}
```

- **Util**

  The Util is a helper class that contains references like GCM ID, app version, username, email etc.

  Code

```
public class Util {
    public static final String EXTRA_MESSAGE = "message";
    public static final String PROPERTY_REG_ID = "registration_id";
    public static final String PROPERTY_APP_VERSION = "appVersion";
    public static final String EMAIL = "email";
    public static final String USER_NAME = "user_name";
    public final static int PLAY_SERVICES_RESOLUTION_REQUEST = 9000;
    public final static String SENDER_ID = "209690052574";
    public static String base_url = "http://192.168.1.33/gcm_demo/";
    public final static String  register_url=base_url+"register.php";
    public final static String  send_chat_url=base_url+"sendChatmessage.php";
}
```

10. Navigate to **app/res/layout**. Create **activity_chat.xml** and **chat_view.xml**. And add the following in the respective file:

10.1 activity_main.xml

This is the main and initial xml file to display. The activity_main.xml file has two textboxes and one login button. The UI is designed to display in Relative Layout.

Code and Description

```
<EditText
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:id="@+id/editText_user_name"
  android:hint="UserName"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="87dp" />
```

The above code creates a textbox of width fill parent, of height wrap content, id set to **editText_user_name**. The text box is displayed on the top and centre horizontal.

```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText_email"
    android:hint="Email"
    android:layout_below="@+id/editText_user_name"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp" />
```

The above code creates a textbox of width fill parent, of height wrap content, id set to **editText_email**. The textbox is displayed below **editText_user_name** textbox and centre horizontal.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Login"
    android:id="@+id/button_login"
    android:layout_below="@+id/editText_email"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="75dp" />
```

The above code creates a button of width and height to wrap the content. Button text is set to Login, id set to **button_login**. The button is displayed below **editText_email** textbox and centre horizontal.

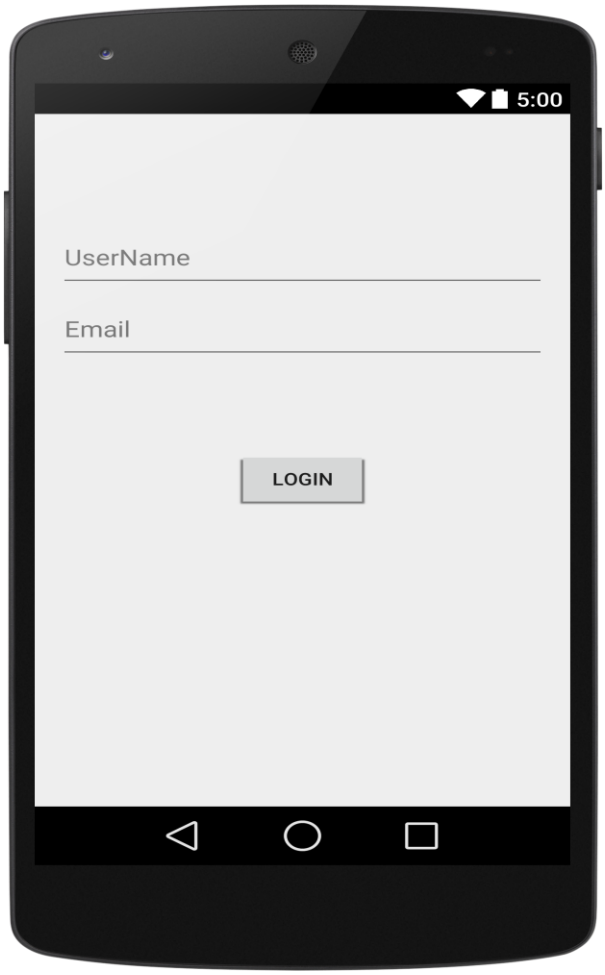


Figure 22: Preview of activity_main.xml

activity_main.xml - Notepad

File Edit Format View Help

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editText_user_name"
        android:hint="UserName"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="87dp" />

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editText_email"
        android:hint="Email"
        android:layout_below="@+id/editText_user_name"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login"
        android:id="@+id/button_login"
        android:layout_below="@+id/editText_email"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="75dp" />

</RelativeLayout>
```

Figure 23: activity_main.xml file

10.2 activity_chat.xml

On successful login, this layout is displayed to the user. This layout contains a list view, two textboxes and a button.

Code and Description

<EditText

android:layout_width="fill_parent"

android:layout_height="wrap_content"

android:id="@+id/editText_mail_id"

android:hint="Enter Reciever mail id"

android:layout_alignParentTop="true"

```
    android:layout_centerHorizontal="true" />
```

The above mentioned code creates a textbox of width set to fill parent, height set to wrap content, id set to **editText_mail_id**, hint set to **Enter Receiver mail id**. The textbox is displayed centre horizontal.

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText_chat_message"
    android:hint="Enter Some thing"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_toLeftOf="@+id/button_send_chat"
    android:layout_toStartOf="@+id/button_send_chat" />
```

The above mentioned code creates a textbox of width and height set to wrap content, id set to **editText_chat_message**, hint set to **Enter Some thing**. The textbox is displayed to the left of **button_send_chat** and to start of **button_send_chat**.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send"
    android:id="@+id/button_send_chat"
    android:singleLine="true"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
     />
```

The above code creates a button of width and height set to wrap content, text set to **Send** and id set to **button_send_chat**. The button is displayed at bottom right.

```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView_chat_messages"
    android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentStart="true"
    android:layout_below="@+id/editText_mail_id"
    android:layout_above="@+id/editText_chat_message"
    android:background="@android:color/transparent"
    android:divider="@null"
    android:dividerHeight="0dp"
    android:scrollbars="none"
    android:stackFromBottom="true"
    android:transcriptMode="alwaysScroll"
    />
```

The above mentioned code displays the list view. The width and height of the list view is set to wrap content, id is set to **listView_chat_messages**. This layout is below **editText_mail_id** textbox and above **editText_chat_message** textbox. The background of the layout is set to transparent.

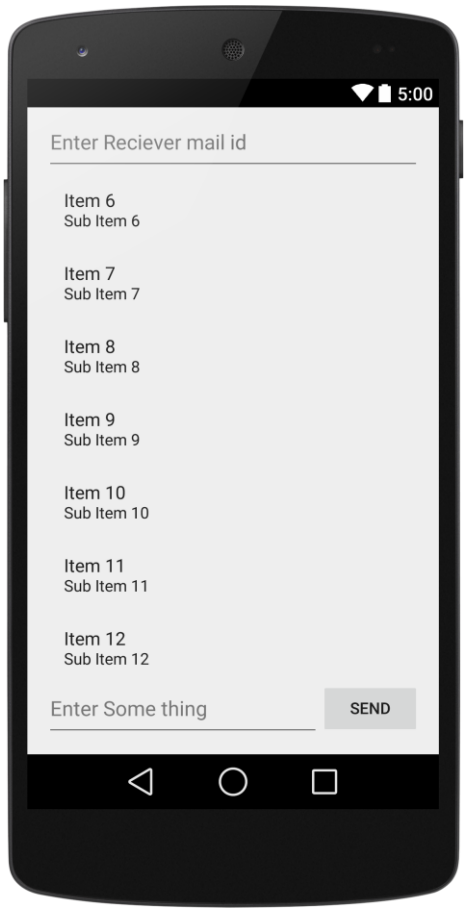


Figure 24: Preview of activity_chat.xml

activity_chat.xml - Notepad

File Edit Format View Help

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="com.example.dell.myapplication.ChatActivity">


    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editText_mail_id"
        android:hint="Enter Reciever mail id"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText_chat_message"
        android:hint="Enter Some thing"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_toLeftOf="@+id/button_send_chat"
        android:layout_toStartOf="@+id/button_send_chat" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send"
        android:id="@+id/button_send_chat"
        android:singleLine="true"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
         />

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView_chat_messages"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:hint="Enter Reciever mail id"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText_chat_message"
        android:hint="Enter Some thing"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_toLeftOf="@+id/button_send_chat"
        android:layout_toStartOf="@+id/button_send_chat" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send"
        android:id="@+id/button_send_chat"
        android:singleLine="true"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
         />

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView_chat_messages"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/editText_mail_id"
        android:layout_above="@+id/editText_chat_message"
        android:background="@android:color/transparent"
        android:divider="@null"
        android:dividerHeight="0dp"
        android:scrollbars="none"
        android:stackFromBottom="true"
        android:transcriptMode="alwaysScroll"
        />


</RelativeLayout>
```

Figure 25: activity_chat.xml file

10.3 chat_view.xml

This layout has two text views.

Code and Description

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Hi"
    android:id="@+id/textView_left_chat"
    android:layout_alignParentTop="true"
    android:padding="2dp"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="42dp"
    android:layout_marginStart="42dp"
    android:layout_marginRight="20dp"
    android:gravity="center"
    android:layout_marginTop="10dp" />
```

The above mentioned code displays a text view. The height and width is set to wrap content. The text appearance is set to **textAppearanceMedium**, text set to **Hi**, id is set to **textView_left_chat**. This textview is displayed at top left and the gravity set to centre.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Hi"
    android:padding="2dp"
    android:id="@+id/textView_right_chat"
    android:layout_alignParentTop="true"
    android:visibility="visible"
```

```
    android:gravity="center"
    android:layout_alignParentRight="true"
    android:layout_marginRight="42dp"
  android:layout_marginLeft="20dp"
  android:layout_marginTop="10dp" />
```

The above code displays the textview. The height and width set to wrap content. The text appearance is set to textAppearanceMedium, text set to Hi, id set to textView_right_chat. This view is aligned to top right and displayed at the centre.

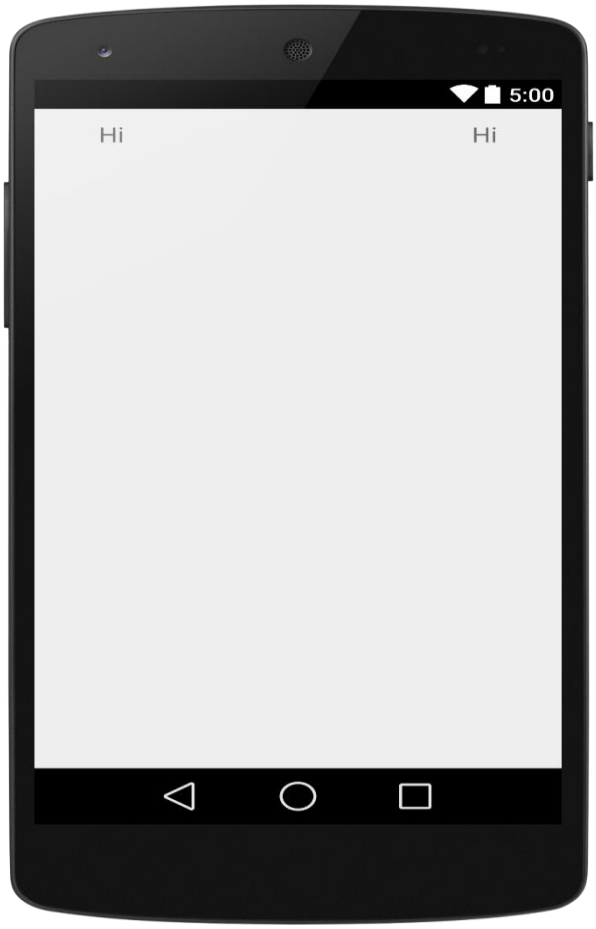


Figure 26: Preview of chat_view.xml

chat_view.xml - Notepad

File Edit Format View Help

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">


    <!--suppress AndroidDomInspection -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Hi"
        android:id="@+id/textView_left_chat"
        android:layout_alignParentTop="true"
        android:padding="2dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="42dp"
        android:layout_marginStart="42dp"
        android:layout_marginRight="20dp"
        android:gravity="center"

        android:layout_marginTop="10dp" />
    <!--suppress AndroidDomInspection -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Hi"
        android:padding="2dp"
        android:id="@+id/textView_right_chat"
        android:layout_alignParentTop="true"
        android:visibility="visible"
        android:gravity="center"

        android:layout_alignParentRight="true"
        android:layout_marginRight="42dp"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp" />


    </RelativeLayout>
```

Figure 27:chat_view.xml file