SETTING UP FCM (FIREBASE CLOUD MESSAGING) ON THE SERVER SIDE USING JAVA

Viraj Rai

July 2016

CONTENTS

INTRODUCTION

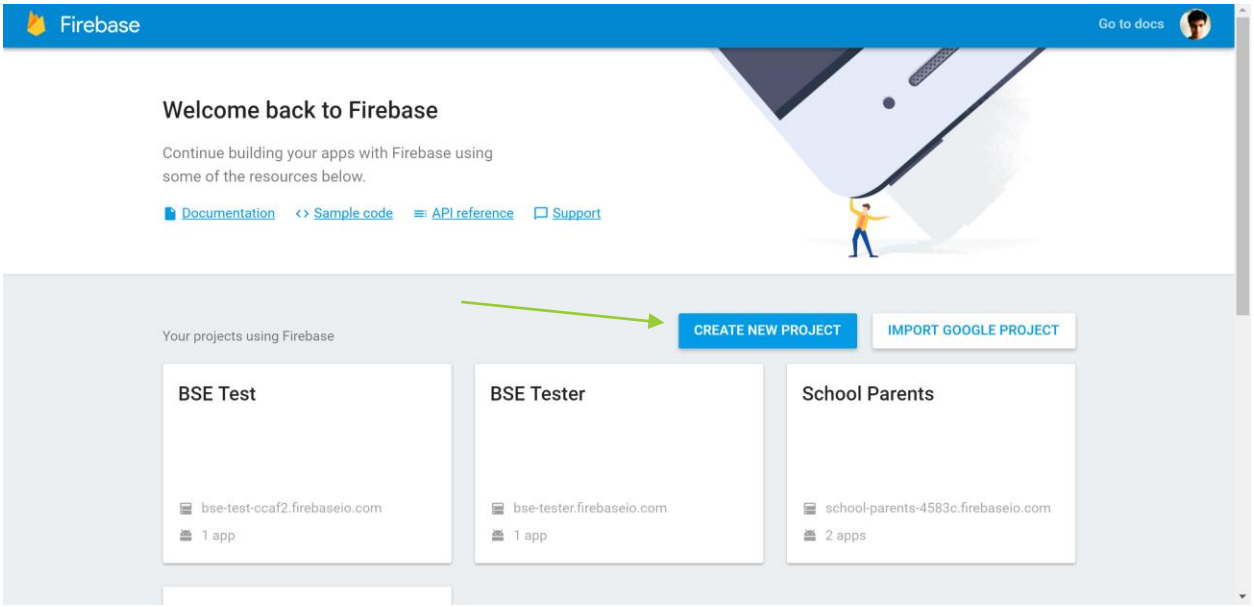
This documentation will guide you to develop a server side implementation of sending push notifications to recipients using Firebase Cloud Messaging. For this, you need to first ensure that you have successfully implemented the client side service for the same. This includes receiving the token number from the client devices and storing them in your server. These are essential as without the token number of a particular device, the server will not be able to send a push notification to that device.

RETRIEVING API KEY AND PROJECT ID FROM FIREBASE CONSOLE

This instruction set will help you retrieve the unique API key and Project ID of your project. These are very important because the Google server recognizes and authenticates the requests made by the server. If you have already created a project, please skip to Step 3.

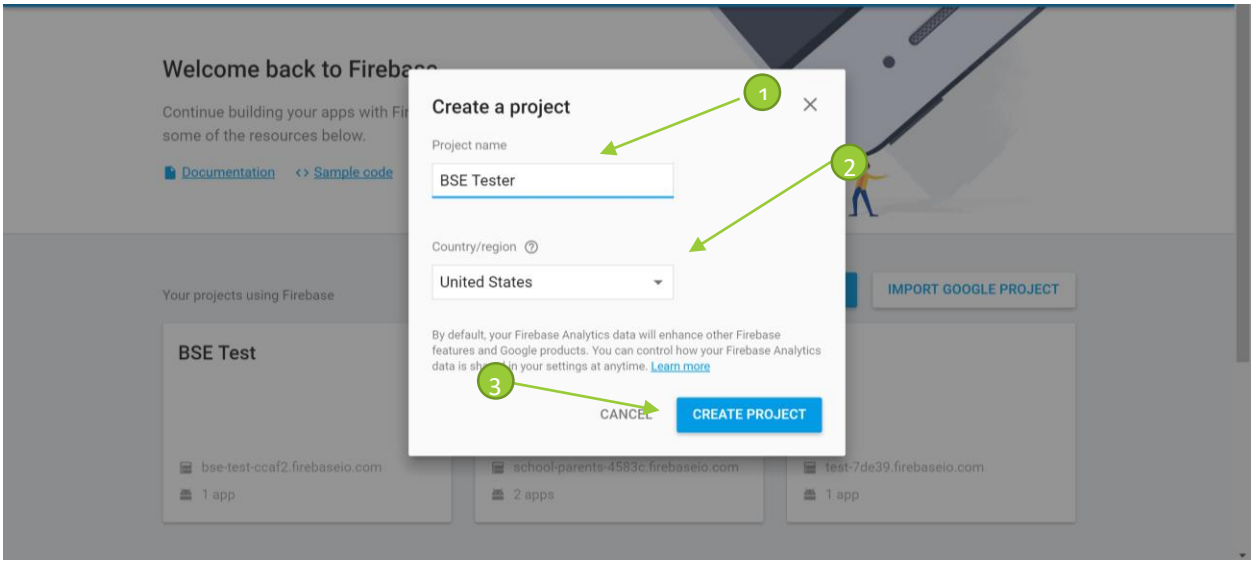
STEP 1

Go to Firebase (https://console.firebase.google.com/). Login in using your Google account, or create a new Account if you do not have one. Once logged in, you will see a screen like the one below. Click on “Create New Project”.



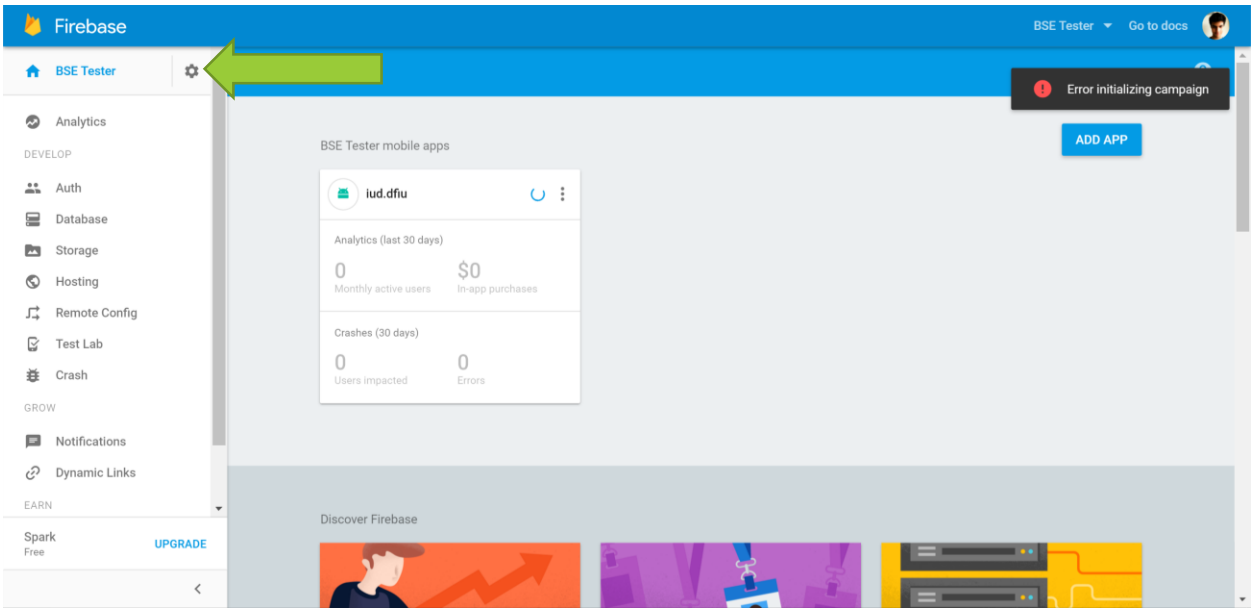
STEP 2

Enter your Project Name and Country and Click “Create Project”.



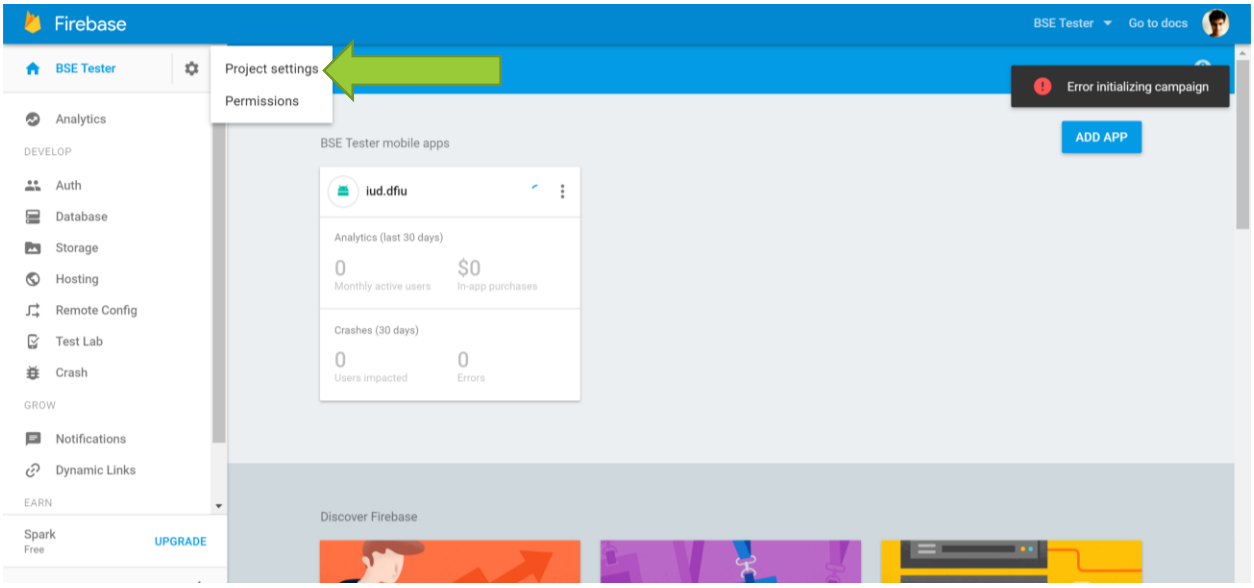
STEP 3

Click on the cog placed on the top left corner of your screen, right next to the project name.



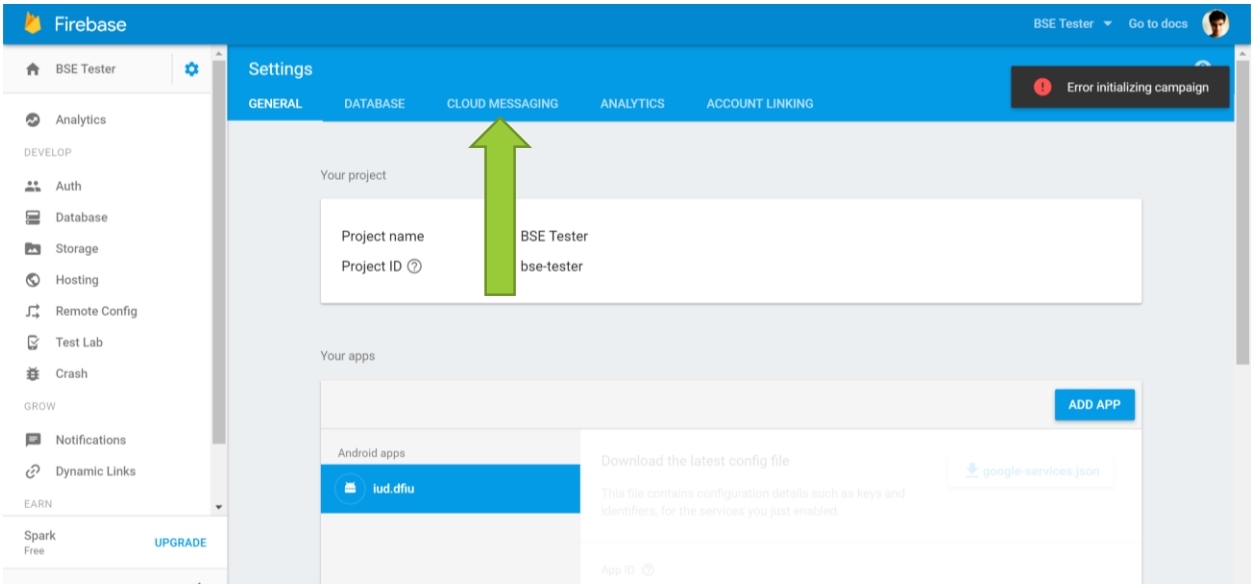
STEP 4

Click on Project Settings



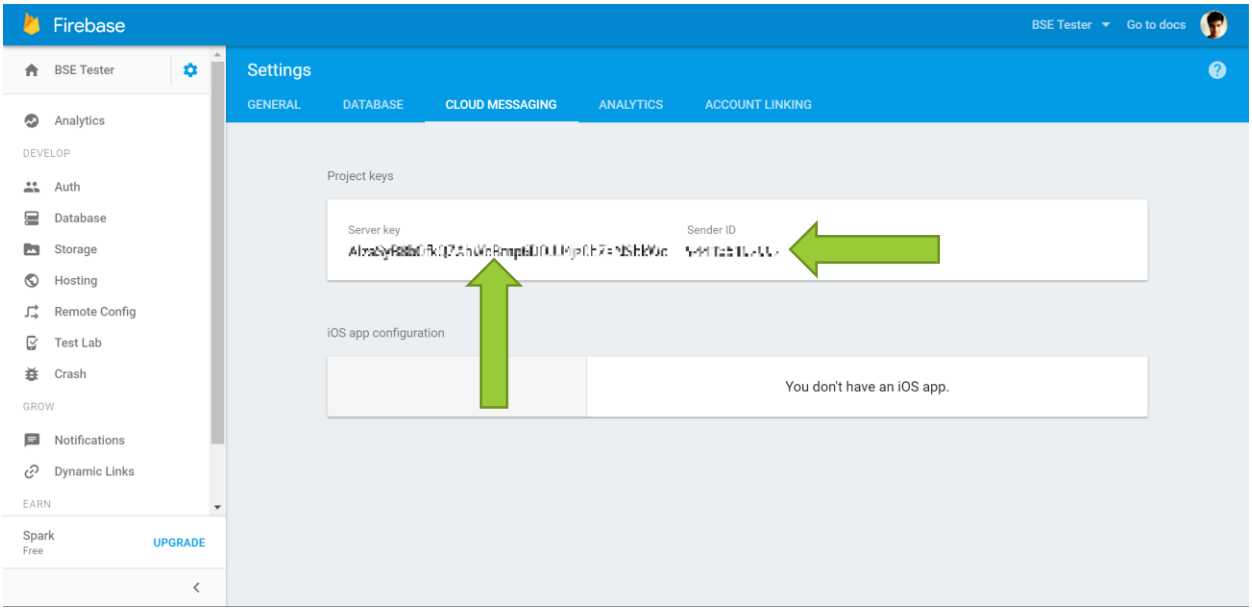
STEP 5

Next, click on Cloud Messaging as we are implementing Firebase Cloud Messaging.



STEP 6

Retrieve the Server Key and Sender ID. These values will be used in your App_Constants class described later.



IMPLEMENTING CLASSES IN PROJECT

STEP 1: ESTABLISHING APP_CONSTANTS

Create a class called App_Constants which will store all information which needs to be available to multiple classes. This information includes the server API key, Project ID, and already created groups. Copy the following code into your App_Constants class and update the API key and Project ID as configured by Firebase.

```
package viraj.com.bse;

import org.json.JSONArray;
import org.json.JSONException;

import java.util.ArrayList;
/**
 * This class contains all the variables required for sending HTTP Posts
 */

public class App_Constants {
    static String PROJECT_ID = "(Enter Sender ID here) ";
    static String API_KEY = "key=(Enter Server Key here)";
    static ArrayList<String> all_tokens = new ArrayList<String>();
    static ArrayList<notificationGroup> groups = new ArrayList<>();


    /**
     * Adds the token to the tokens ArrayList if the token is not already present
     * @param token the token to be added to the databse
     */

    public static void add_token(String token){
        if (!all_tokens.contains(token)) all_tokens.add(token);
    }

    /**
     * Adds a new group to the JSON along with its notification_key
     * @param name Name of the group
     * @param notification_key The notification key of the respective group as received from the
     *                         Google server
     */
    public  static void new_group(String name, String notification_key, JSONArray registration_ids){

        if (!notificationGroup.hasName(name)){
            notificationGroup temp_group = new notificationGroup(name,notification_key,registration_ids);
            groups.add(temp_group);
        }
    }
```

```
    /**
     * This group adds tokens tokens to an already existing group
     * @param name Name of the group
     * @param notification_key Notification key of the group (used for validation)
     * @param registration_ids JSON Array of registration ids to be added to the group
     */
    public static void addToGroup(String name, String notification_key, JSONArray registration_ids){
        //Check if the group exists
        if (notificationGroup.hasName(name)){

            //Iterate through all groups and check if the group exists and the notification key is
            //valid
            for (notificationGroup i : groups){
                if (i.toString().equals(name) && i.getKey().equals(notification_key)){
                    //add tokens to the particular group
                    i.add_tokens(registration_ids);
                    addToAll(registration_ids);
                }
            }
        }
    }

    /**
     * Add tokens to the Master ArrayList containing all tokens if they are already not there
     * @param tokens A JSONArray of tokens
     */

    public static void addToAll(JSONArray tokens){
        try {
            for (int i = 0; i < tokens.length(); i++) {
                if (!all_tokens.contains(tokens.getString(i))) {
                    all tokens.add(tokens.getString(i));
                }
            }
        }
        catch (JSONException e){
            e.printStackTrace();
        }
    }

    /**
     * Returns the notification key of a group
     * @param name Name of the group whose notification key is required
     * @return Notification key as a string if a valid group name was passed, else returns "error"
     */
    public static String getNotificationKey(String name){
        if (notificationGroup.hasName(name)){
            for(notificationGroup i : groups){
                if (i.toString().equals(name)){
                    return i.getKey();
                }
            }
        }
        return "error";
    }

}
```

STEP 2: CREATING CLASS TO ORGANISE GROUPS

Create a class called notificationGroup. This is class acts as a custom data type which stores the information of a single notification group. This includes the name, the notification key and the tokens registered with it.

```
import org.json.JSONArray;
import org.json.JSONException;
import java.util.ArrayList;

/**
 * This class acts as a custom data type which helps store a notification group on the server.
 * This includes the groups name, notification key and the registered tokens.
 */
public class notificationGroup {
    String name;
    String key;
    JSONArray members;
    static ArrayList<String> allNames = new ArrayList<String>();

    /**
     * Constructor takes in information about the group to store a new group
     * @param name Name of the group
     * @param key Notification key of the group as received by the server
     * @param tokens A JSONArray of all the tokens already registered with the group
     */
    public notificationGroup(String name, String key, JSONArray tokens){
        members = tokens;
        this.name = name;
        this.key = key;
        allNames.add(name);
    }

    /**
     *
     * @return Returns the name of the group
     */
    public String toString(){
        return name;
    }


    /**
     *
     * @return The notification key of the group
     */
    public String getKey(){
        return key;
    }

    /**
     *
     * @return All the tokens registered with the group in a JSON Array
     */
    public JSONArray getTokens(){
        return members;
    }

    /**
     * Static class which searches all groups to check whether a group name already exists.
     * @param name The group name
     * @return True if group name already exists, else returns False.
     */
    public static boolean hasName(String name){
        return allNames.contains(name);
    }

    /**
     * Adds tokens to the list of registered tokens
     * @param tokens Tokens to be added to the list of registered tokens
     */
    public void add tokens(JSONArray tokens){
        for (int i = 0; i < tokens.length();i++)
            try {
                members.put(tokens.getString(i));
            } catch (JSONException e) {
                e.printStackTrace();
            }
    }
}
```

STEP 3: CREATING CLASS TO SEND HTTP POST

Create a class called sendHTTPost which will enable us to send the HTTP POST to the Google server, which will in turn send the push notifications. This class is an Async Task, which means it will run separate from the main thread (think of it as a background process). It accepts multiple arguments of type JSONObject and stores them in the variable params in the same order in which the arguments were passed. This code is configured to pass only one argument, which is the payload of the push notification. For more information regarding what is a payload, and what it consists of, please click here. It is crucial that the payload contains either the *to:* field or the *registration_ids:* field, which instructs the server which device(s) to send the notification to.

```
import android.os.AsyncTask;
import android.util.Log;
import org.json.JSONObject;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Scanner;

/**
 * Send HTTP post directly with one or many users, using their respective tokens.
 * It is an async task seperate from the main thread.
 */

public class sendHTTPPost  extends AsyncTask<JSONObject,Void,Void>{
    JSONObject mainJson = new JSONObject();
    String response;
    String TESTING = App_Constants.TESTING;
    boolean post sent;
    Context context;


    /**
     *Contains the algorithm to be executed asynchronously
     * @param params Is an array of JSON Objects. The first item is a JSON object containing data
     *               of the HTTP post such as who it is targetted to, its notification title and
     *               body, and its data message. It also contains other parameters of the payload.
     * @return Void
     */

    @Override
    public  Void doInBackground(JSONObject... params){
        try {
            mainJson = params[0];
            URL url  = new URL("https://fcm.googleapis.com/fcm/send");
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            conn.setDoOutput(true);
            conn.setRequestProperty("Content-type","application/json");
            conn.setRequestProperty("Authorization",App_Constants.API_KEY);
            Log.d(TESTING,"Property set");

            conn.getOutputStream().write(mainJson.toString().getBytes("UTF-8"));

            // Extract the input stream if the group creation was successful, else extract the error
            // message
            InputStream is;
            if (conn.getResponseCode() >= HttpURLConnection.HTTP_BAD_REQUEST){
                is = conn.getErrorStream();
            }
            else {
                is = conn.getInputStream();
            }
            response = new Scanner(is, "UTF-8").useDelimiter("\\A").next();
            is.close();
            Log.d(TESTING,"response: "+ response);


            if (conn.getResponseCode()==200) post_sent = true;
            else post_sent = false;
```

```
        } catch (MalformedURLException e) {
            Log.d(TESTING,"Malformed URL");
        } catch (IOException e){
            Log.d(TESTING,"IO exception");
        }
        catch (Exception e){
            Log.d(TESTING,e.getClass().toString());
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public void onPostExecute(Void result){
        if (post_sent) {
            Log.d(TESTING,"sendHTTPPost Successful");
            Toast.makeText(context, "Message sent successfully", Toast.LENGTH_SHORT).show();
        }
        else Log.d(TESTING,"Error in executing sendHTTPPost");
    }

}
```

STEP 4: CREATING CLASS TO MAKE NEW GROUPS

Create a createGroup class. This class is a Async Task and enables us to create new notification groups with the Google server. Creating notification groups is more efficient as the server does not have to constantly passing thousands of tokens with each and every HTTP Post request and send multiple HTTP Post requests to send a mass push notification. Instead, the server just sends the group notification key (which is provided by the Google server when the group is created) in the to field of the payload and the notification is automatically sent to all the devices registered in the group.

This class accepts multiple arguments of type String Array and stores them in the variable params as per the order in which the arguments were passed. This code is configured to pass only 2 arguments to params. The first item (index 0) is a String Array containing only 1 item, the name of the new group. The second item (index 1) is a String Array containing all the registration ids to be registered with the new group. If the group is created successfully, the information of this new notification group is stored through the new_group method of the App_Constants class. The information stored is the group name, its notification key, and the token ids registered with this group.

```
import android.app.ProgressDialog;
import android.content.Context;
import android.os.AsyncTask;
import android.util.Log;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Scanner;
```

```java
/**
 *This class is a Async Task and enables us to create new notification groups with the Google
 * server. It is very useful as instead of constantly passing thousands of tokens with each and
 * every HTTP Post request to send a mass push notification, the server can just send the group
 * notification key (which is provided by the Google server when the group is created) in the to
 * field of the payload and the notification is automatically sent to all the devices registered
 * in the group.
 */
public class createGroup  extends AsyncTask<String[],Void,JSONObject>{
    JSONObject mainJson = new JSONObject();
    String response;
    String TESTING = App Constants.TESTING;
    String notif key return;
    boolean success;
    String name;
    JSONObject send_info = new JSONObject();
    ProgressDialog progressDialog;
    Context context;
    public boolean groupCreated;

    /**
     * Contains the algorithm to be executed asynchronously
     * @param params It is an array of String Array.
     *               The first item in the array is (note) an array containing a single element
     *               which is the name of the new group. The second element is the array of
     *               registration ids (tokens) to be added to this new group.
     *
     * @return Notification key, if the group was successfully mane, or the error message received
     *         by the server if the group was not created. This is passed on the the onPostExecute
     *         method.
     */
    @Override
    public  JSONObject doInBackground(String[]... params){
        JSONObject return_JSON = new JSONObject();
        try {
            name =  params[0][0];
            //Creates the payload JSON which will be sent to create the group
            mainJson.put("operation", "create");
            mainJson.put("notification key name", name);
            mainJson.put("registration ids",new JSONArray(params[1]));
            mainJson.put("project_id",App_Constants.PROJECT_ID);

            return_JSON.put("registration_ids",new JSONArray(params[1]));

            //Initiates the connection with the Google server
            URL url  = new URL("https://android.googleapis.com/gcm/notification");
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            //Sets the headers of the HTTP Post
            conn.setDoOutput(true);
            conn.setRequestProperty("Content-type","application/json");
            conn.setRequestProperty("Authorization",App Constants.API KEY);
            conn.setRequestProperty("project id",App Constants.PROJECT ID);


            //Send the payload to the Google server
            conn.getOutputStream().write(mainJson.toString().getBytes("UTF-8"));
            InputStream is;

            // Extract the input stream if the group creation was successful, else extract the error
            // message
            if (conn.getResponseCode() >= HttpURLConnection.HTTP_BAD_REQUEST){
                is = conn.getErrorStream();
            }
            else {
                 is = conn.getInputStream();
            }
            response = new Scanner(is, "UTF-8").useDelimiter("\\A").next();
            is.close();
            Log.d(TESTING,"response: "+ response);

            //convert the response into a JSON object
            JSONObject response_JSON = new JSONObject(response);
```

```
            // Return the response key if available, otherwise return the error code
            if (response_JSON.has("notification_key")){
                Log.d(TESTING,"{name: " + response_JSON.getString("notification_key"));
                return_JSON.put("result",response_JSON.getString("notification_key"));
                return return_JSON;
            }
            else {
                Log.d(TESTING, "{name: error");
                return_JSON.put("result","error");
                return return_JSON;
            }
        }


        catch (JSONException e){
            Log.d(TESTING,"JSON Exception");
        }
        catch (MalformedURLException e) {
            Log.d(TESTING,"Malformed URL");
        } catch (IOException e) {
            Log.d(TESTING,"IO exception");
            e.printStackTrace();
        }
        catch (Exception e){
            Log.d(TESTING,e.getClass().toString());
            e.printStackTrace();
        }

        return null;

    }

    /**
     * This method stores the group information in the App_Constants as an established group if the
     * group was created successfully.
     * @param result it is a JSONObject containing 2 fields. It stores a String result, and a
     *               JSONArray of all the tokens registered to this new group. If the group is
     *               created successfully, result field will contain the group's notification_key,
     *               else it will contain "error".
     */
    @Override
    public void onPostExecute(JSONObject result){
        try {
            if (result != null) {
                notif_key return = result.getString("result");

                if (notif_key_return.equals("error")) {
                    success = false;
                } else {
                    success = true;
                }
                Log.d(TESTING,"success = " + success);

                if (success) {
                    // Store the group information
                    App_Constants.new_group(name, notif_key_return, result.getJSONArray("regitration_ids"));
                    groupCreated=true;
                    Toast.makeText(context, "Group created successfully", Toast.LENGTH_SHORT).show();
                }
            }
            else Log.d(TESTING, "Post exec failed");
        }
        catch (JSONException e){
            e.printStackTrace();
        }
    }
}
```

STEP 5: CREATING CLASS TO ADD TOKENS TO AN EXISTING GROUP

Create an addToGroup class. This class extends Async Task and uses HTTP Post to add tokens to already created groups. This group accepts multiple arguments of type String Array. The first argument is a String array with one element, the name of the group. The second argument is a String array of all the token ids needed to be added to the group. The class automatically extracts the notification_key from the App_Constants class.

After the new tokens are added, the group information is updated and the new tokens are stored. The class also adds the tokens to the master ArrayList containing all tokens if any token was not already registered.

```
import android.os.AsyncTask;
import android.util.Log;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Scanner;

/**
 * This class helps register new token ids to an existing group
 */
public class addToGroup  extends AsyncTask<String[],Void,JSONObject>{
    JSONObject mainJson = new JSONObject();
    String response;
    String TESTING = App_Constants.TESTING;
    String notif_key_return;
    boolean success;
    String name;
    JSONObject return_JSON = new JSONObject();

    /**
     * Takes in multiple arguments of String Arrays
     * @param params First index is group name, second index is an Array of tokens needed to be
     *               registered on the group
     * @return JSON Object with newly registered tokens, and the response received fro the HTTP Post
     */
    @Override
    public  JSONObject doInBackground(String[]... params){
        try {
            //Ensure the group exists
            if (App_Constants.getNotificationKey(name).equals("error")) throw new Exception();
            name = params[0][0];
            mainJson.put("operation", "add");
            mainJson.put("notification key name", name);
            mainJson.put("notification key",App Constants.getNotificationKey(name));
            mainJson.put("registration_ids",new JSONArray(params[1]));

            return_JSON.put("registration_ids",new JSONArray(params[1]));

            URL url  = new URL("https://android.googleapis.com/gcm/notification");
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            conn.setDoOutput(true);
            conn.setRequestProperty("Content-type","application/json");
            conn.setRequestProperty("Authorization",App_Constants.API_KEY);
            conn.setRequestProperty("project_id",App_Constants.PROJECT_ID);


            conn.getOutputStream().write(mainJson.toString().getBytes("UTF-8"));
            InputStream is;

            if (conn.getResponseCode() >= HttpURLConnection.HTTP_BAD_REQUEST){
                is = conn.getErrorStream();
            }
            else {
                is = conn.getInputStream();
            }
            response = new Scanner(is, "UTF-8").useDelimiter("\\A").next();
            is.close();

            Log.d(TESTING,"response: "+ response);
```

```
            JSONObject response_JSON = new JSONObject(response);

            if (response_JSON.has("notification_key")){
                Log.d(TESTING,"{name: " + response_JSON.getString("notification_key"));
                return JSON.put("result", response JSON.getString("notification key"));
                return return JSON;
            }
            else {
                Log.d(TESTING, "{name: error");
                return_JSON.put("result","error");
                return return JSON;
            }
        }
        catch (JSONException e){
            Log.d(TESTING,"JSON Exception");
        }
        catch (MalformedURLException e) {
            Log.d(TESTING,"Malformed URL");
        } catch (IOException e) {
            Log.d(TESTING,"IO exception");
            e.printStackTrace();
        }
        catch (Exception e){
            Log.d(TESTING,e.getClass().toString());
            e.printStackTrace();
        }

        return null;
    }

    /**
     * Adds the newly registered tokens to the respective group if the process was successful
     * @param result A JSON Object containing a JSON Array with of newly added token ids and
     *               a String containing the notification key if the process was successful,
     *               or "error" if there was an error in adding the keys to the group.
     */
    @Override
    public void onPostExecute(JSONObject result){
        try {
            if (result != null) {
                notif_key_return = result.getString("result");

                if (notif_key_return.equals("error"))  success = false;
                else success = true;

                if (success)  App_Constants.addToGroup(name, notif_key_return,
result.getJSONArray("registration_ids"));
            }
        }
        catch (JSONException e){
            e.printStackTrace();
        }
    }

}
```

PAYLOAD OF PUSH NOTIFICATION

This will be a method in your main method which will help create a JSON Object to be sent with your HTTP Post. This will contain all the data of the notification such as data message, notification title and body, any click_action along with any custom fields you want to send with the notification. This method does not add the *to:* field to the JSON. This field must be specified at the time of execution of the HTTP Post. This method is only intended to generate data of a push notification and not target them. For more information regarding payload, click here. For more information regarding the payload fields, click here.

Note: You can alter the method so it accepts arguments and creates a payload as per your needs.

CREATING PAYLOAD

```
/**
 * Helps create the payload which will be sent via HTTP Post. This will contain all the data of
 * the notification such as data message, notification title and body, any click action along
 * with any custom fields you want to send with the notification. This method does not add the
 * to: field to the JSON. This field must be specified at the time of execution of the HTTP
 * Post. This method is only intended to generate data of a push notification and not target
 * them.
 *
 * @return Returns a JSON Object which acts as the payload for the HTTP Post
 */

public JSONObject payloadBuilder(){
    JSONObject data = new JSONObject();
    JSONObject notification = new JSONObject();
    JSONObject send_JSON = new JSONObject();

    try {

        data.put("message", "This is the test data message");
        notification.put("title", "Test Title");
        notification.put("body", "Test Notification Body");
        send_JSON.put("data", data);
        send_JSON.put("notification", notification);
        return send_JSON;

    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
```

IMPLEMENTATION OF CLASSES

SENDING A NOTIFICATION TO A SINGLE USER

```
String name;
try {
    EditText editText = (EditText) findViewById(R.id.(Edit text name));
    name = editText.getText().toString();

    if (App Constants.all tokens.contains(name)) {
        JSONObject send_JSON = payloadBuilder();
        send_JSON.put("to",name);

        send_POST = new sendHTTPPost();
        send_POST.context = MainActivity.this
        send POST.execute(send JSON);


    }
    else{
        Toast.makeText(MainActivity.this, "Group does not exist", Toast.LENGTH_SHORT).show();
    }
} catch (JSONException e) {
    e.printStackTrace();
}
```

SENDING A NOTIFICATION TO AN EXISTING GROUP

```
String name;
try {
    EditText editText = (EditText) findViewById(R.id.(Edit text name));
    name = editText.getText().toString();

    if (!App Constants.getNotificationKey(name).equals("error")) {
        JSONObject send_JSON = payloadBuilder();
        send_JSON.put("to", App_Constants.getNotificationKey(name));

        send_POST = new sendHTTPPost();
        send_POST.context = MainActivity.this
        send POST.execute(send JSON);


    }
    else{
        Toast.makeText(MainActivity.this, "Group does not exist", Toast.LENGTH_SHORT).show();
    }
} catch (JSONException e) {
    e.printStackTrace();
}
```

SENDING A NOTIFICATION TO MULTIPLE USERS

```
try {
    JSONObject send_JSON = payloadBuilder();
    send_JSON.put("registration_ids", new JSONArray((Array list of all the tokens you want to send the
message));

    sendHTTPPost send_POST = new sendHTTPPost();
    send_POST.context = MainActivity.this;
    send_POST.execute(send_JSON);

} catch (JSONException e) {
    e.printStackTrace();
}
```

CREATE A NEW GROUP

```
try{
        String name = //Your Group Name
        if (!name.equals("") && !notificationGroup.hasName(name)) {
            String[] name_array = {name};
            ArrayList<String> tokens = (An array list of all tokens you want to register with this group);
            if (tokens.size() > 0) {
                String[] tokens array = new String[tokens.size()];
                for (int i = 0; i < tokens.size(); i++) {
                    tokens_array[i] = tokens.get(i);
                }
                createGroup sender = new createGroup();
                sender.context = MainActivity.this;
                sender.execute(name array,tokens array);
                }
        }
        else{
            if (name.equals("")) Toast.makeText(getApplicationContext(),
                "Enter name",Toast.LENGTH_SHORT).show();

            else if (notificationGroup.hasName(name)) Toast.makeText(getApplicationContext(),

                                        "Group already Exists",Toast.LENGTH_SHORT).show();


            else Toast.makeText(MainActivity.this, "Error", Toast.LENGTH_SHORT).show();


        }
    }
    catch ( JSONException e){
        e.printStackTrace();
    }
}
```

ADD TOKENS TO GROUP

```
try{
        String name = //Your Group Name
        if (!name.equals("") && notificationGroup.hasName(name)) {
            String[] name_array = {name};
            ArrayList<String> tokens = (An array list of all tokens you want to register with this group);
            if (tokens.size() > 0) {
                String[] tokens array = new String[tokens.size()];
                for (int i = 0; i < tokens.size(); i++) {
                    tokens_array[i] = tokens.get(i);
                }
                addToGroup sender = new addToGroup();

                sender.execute(name_array,tokens_array);
                }
        }
        else{
            if (name.equals("")) Toast.makeText(getApplicationContext(),

                                    "Enter name",Toast.LENGTH_SHORT).show();


else if (!notificationGroup.hasName(name)) Toast.makeText(getApplicationContext(),

                                        "Group does not Exists",Toast.LENGTH_SHORT).show();


else Toast.makeText(MainActivity.this, "Error", Toast.LENGTH_SHORT).show();


        }
    }
    catch ( JSONException e){
        e.printStackTrace();
    }
}
```