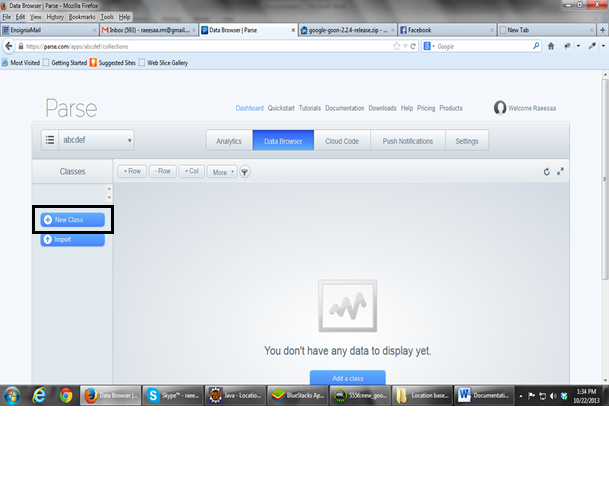
**WORKING WITH PARSE**

Parse.com provides backend services for desktop and mobile applications. They provide SDK for different platforms like Android, iOS, Javascript, Windows Phone 8, Windows 8(.NET SDK) and OSX. We will see Parse API for android.

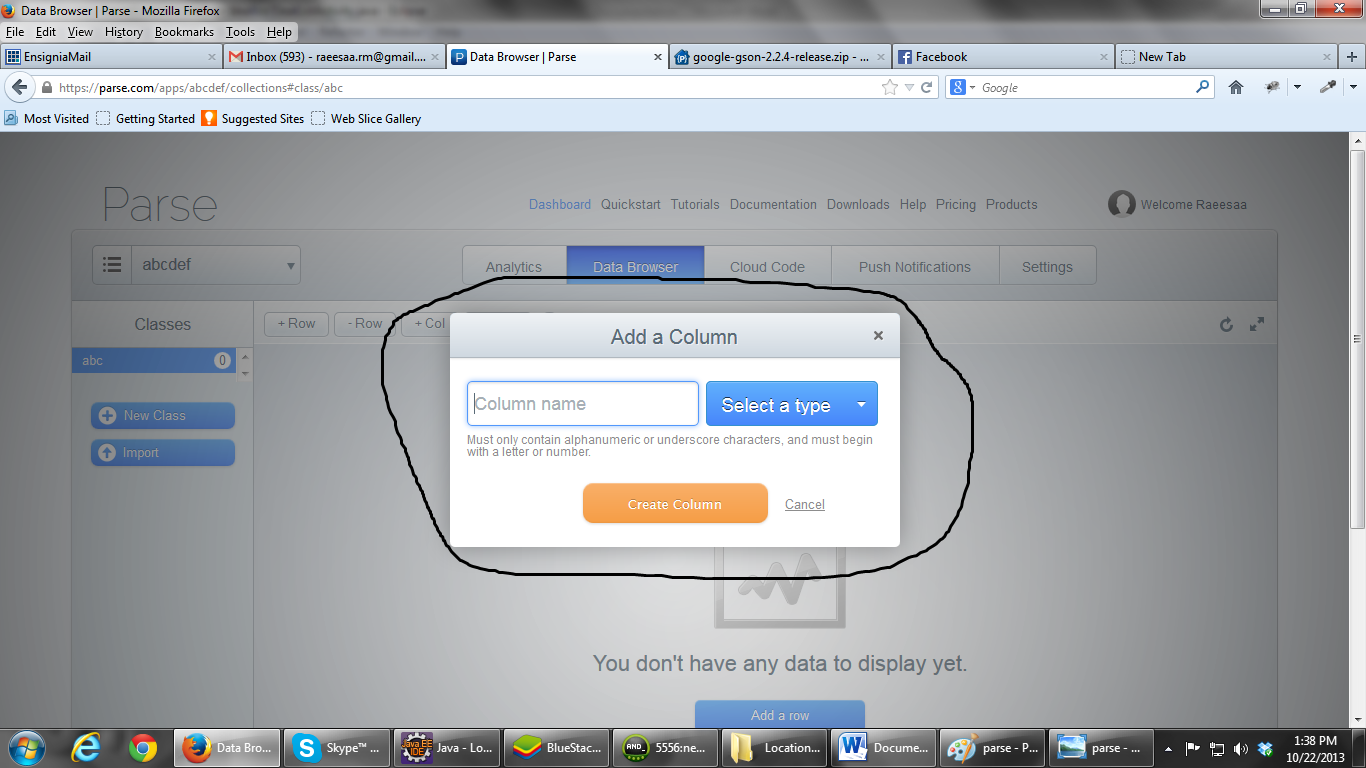
**Creating account, project and database in Parse:**

Following are the steps for creating an account, project and database in parse:

1. Go to Parse.com and sign up.
2. Login to the parse.
3. Click on create new App.
4. Fill the details and Parse will create an Application ID and Client Key for the app.
5. In order to create database tables (i.e. classes) click on ‘Data Browser’ tab. You will get the following screen. Click ‘New Class’ button.



You can add fields by clicking on +Col. Enter the column name and data type in the pop-up that appears after clicking +Col.



Keep on adding all the fields **except primary key. Parse creates a column for primary key by itself called objectId.**

1. Create all the tables. Once the tables/classes are created, we can add/fetch the data from Parse tables using code in our android application. There is a very good and easy to understand documentation of Parse API for android which explains about initializing Parse, saving the data into Parse DB and fetching the data from Parse DB. The link for the documentation is:

[**https://parse.com/docs/android\_guide#objects**](https://parse.com/docs/android_guide#objects)

**SETTING UP APPLICATION FOR USING PARSE:**

Prerequisites:

1. Latest Android SDK.
2. Android Application Project to use Parse as backend for.
3. Parse SDK which can be downloaded from: <https://parse.com/docs/downloads>.

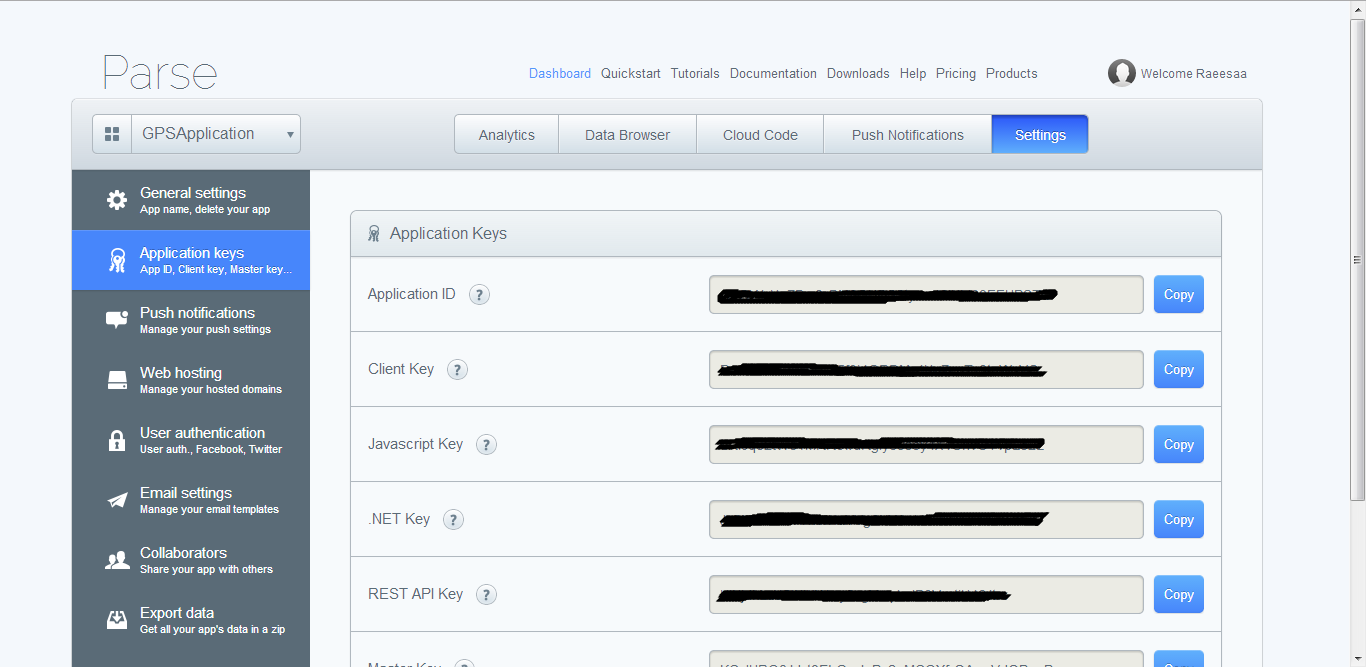
Linking project with Parse:

1. Extract the contents of downloaded Parse SDK zip file into ‘libs’ folder of android application.
2. Initialize Parse in overridden onCreate() method of Application class fo project as follows:

public void onCreate() {

    Parse.initialize(this, "YOUR\_APPLICATION\_ID", "YOUR\_CLIENT\_ID");   
}

Replace “YOUR\_APPLICATION\_ID” and “YOUR\_CLIENT\_KEY” by the application id and client key created for your app by Parse. This application id and client key can be obtained anytime by going into ‘Settings’ section of Parse dashboard.

****

1. Add following permissions in AndroidManifest.xml file just before <application> tag:

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

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

1. In case we need to track the statistics regarding application like how many times application was opened or on how many devices it is installed, include following line in main Activity of your application:

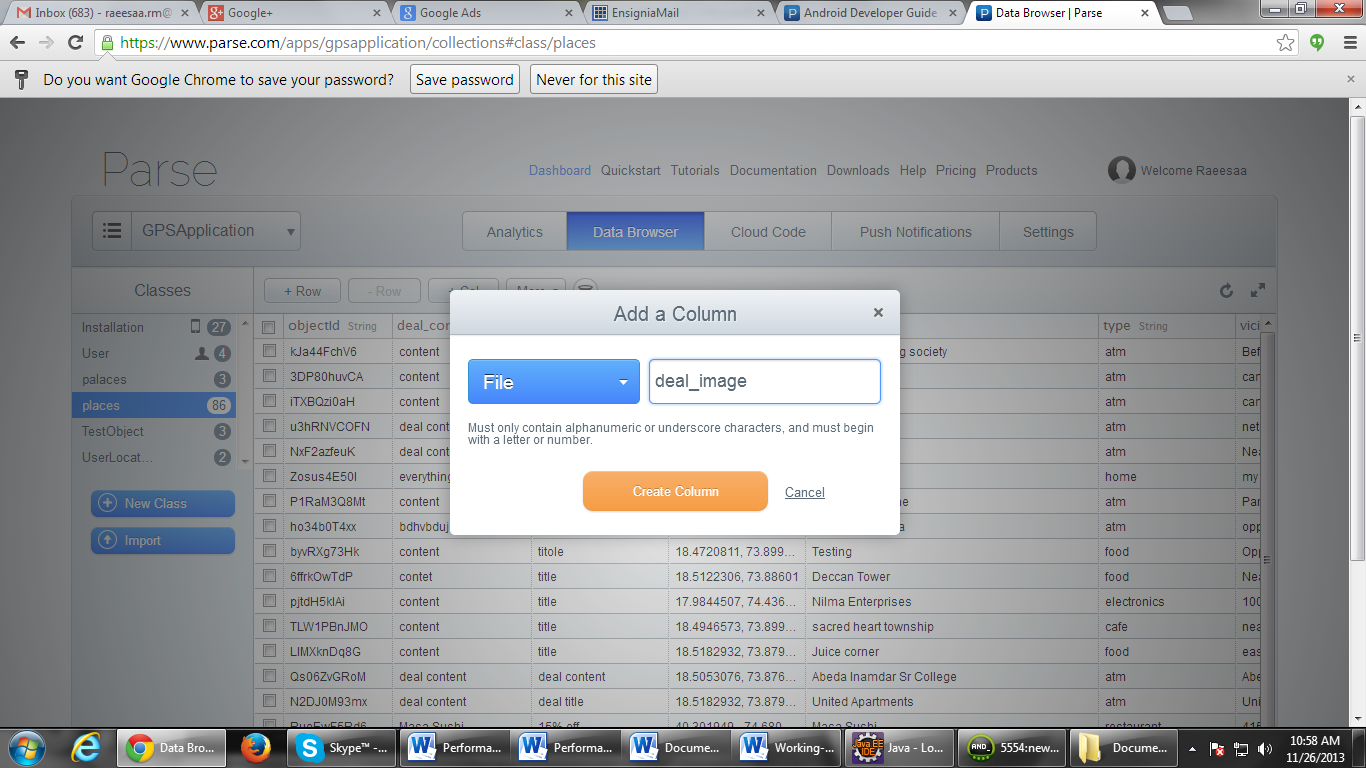
ParseAnalytics.trackAppOpened(getIntent());

All the application statistics can be viewed in Parse dashboard.

**SAVING A FILE IN PARSE:**

Parse API provides a class called ParseFile that lets you store application files like documents, images, videos, etc. in the cloud.

Suppose I have a class(table) called ‘deal’ in Parse having columns (deal\_title, deal\_content) and now I need to store an image related to each deal. For achieving this I need to add a column to ‘deal’ class in parse say ‘deal\_image’ of type ‘File’.



Now we can use ParseFile class of Parse API for saving the image to the Parse cloud. First we need to convert file(image in our case) into byte[] form and then create ParseFile object with it.

InputStream is = **new** FileInputStream(“/storage/sdcard/Download/landscapes.jpg”);

ParseFile file = **new** ParseFile(“img.jpg”, toByteArray(is));

file.save();

The first parameter of ParseFile constructor is file name. The two things to be noticed about this parameter are:

* You don't need to worry about filename collisions. Each upload gets a unique identifier so there's no problem with uploading multiple files named img.jpg.
* It's important that you give a name to the file that has a file extension. This lets Parse figure out the file type and handle it accordingly. So, if you're storing PNG images, make sure your filename ends with .png.

*file.save()* saves the file into cloud synchronously. We can also save it asynchronously using *saveInBackground()*.

After the save completes, we can associate this ParseFile with ParseObject. On our case, we can associate the saved image with corresponding deal. This can be done as below:

ParseObject object = **new** ParseObject("deal");

object.put("deal\_content", deal\_content);

object.put("deal\_title", deal\_title);

object.put("deal\_image", file);

object.saveInBackground();

Adding a ParseFile to ParseObject is similar to adding any other piece of data.

We can retrieve this file back as follows:

ParseQuery<ParseObject> query = **new** ParseQuery<ParseObject>("deal");

Query.whereEqualTo(“deal\_title”,”some deal”);

query.findInBackground(**new** FindCallback<ParseObject>() {

@Override

**public** **void** done(List<ParseObject> objects, ParseException e) {

// **TODO** Auto-generated method stub

**if** (e == **null**) {

**ParseFile image\_file = objects.get(0).getParseFile("deal\_image");**

}

else {

}

});

Once file is retrived from Parse, we can use different methods of ParseFile to access its data. Some of the commonly used methods are:

* getUrl() : Returns the file URL that can be used to access the file
* getData() : Returns the file data in byte[] form
* getDataInBackground() : Returns the file data in byte[] form asynchronously

Snippet for getDataInBackground():

Image\_file.getDataInBackground(new GetDataCallback() {

   public void done(byte[] data, ParseException e) {

     if (e == null) {

       // data has the bytes for the image

     } else {

       // something went wrong

   }

  }

});

Snippet for getUrl():

String url = image\_file.getUrl(); //Returns image url