**The Short Document of “BeLeaf”(Suicide Prevention Application) about Sending & Receiving Data**

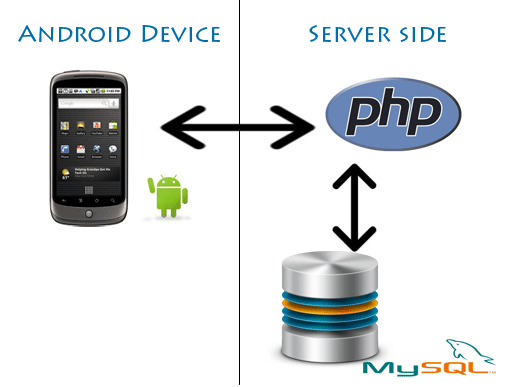
*by*

*Nawat Vatanawood (email:* [*nawatv@gmail.com*](mailto:nawatv@gmail.com)*)*

1. **Overview**

An overview makes the project “BeLeaf” easier to “understand and implement”. In this project, it is divided into two parts:

* 1. System Overview



1. **Android Application**

**2.2 Database**

**2.1 API**

**2. Server Side**

* + 1. Android Application

Android Application is a software application running on the Android platform. It is built from a project by using programming languages such as *Java* and so on. Before being built, The project is compiled by an android IDE. There are a lot of libraries for android application development.

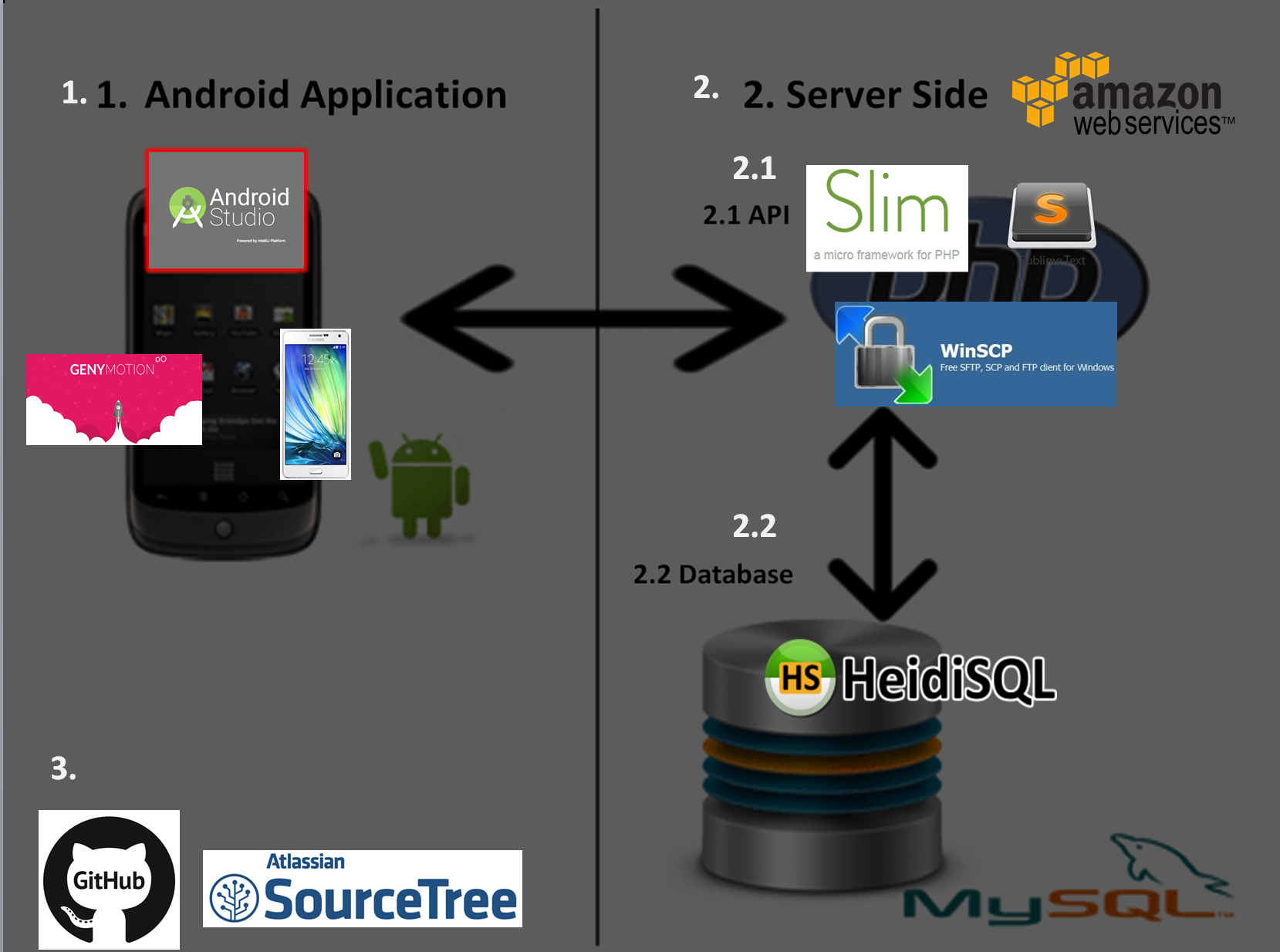
* + 1. Server Side

Server Side refers to operations that are performed by a server. In this project, It is divided into two main parts:

1.1.2.1 API (Php): API stands for “Application Program Interface”. It is a set of routines, protocols, and tools for building software applications. In this project, The API is the intermediate component between Android Application and Database. It is built by using *Php.* For one function, It receives a request sent from the application and queries data in an database. After getting the data, It transforms the data into a response and returns it to the application.

1.1.2.2 Database (MySQL): Database is a collection of information that is organized so that it can easily be accessed, managed, and updated. In this case, database refers to “Database Software” such as *MySQL* and so on.

1.2. Tools Overview



1.2.1 Android Application Tools

Android Studio

(See more, Download & Set up: <https://developer.android.com/studio/index.html>) :

Android studio is an android IDE (Integrated Development Environment). A project created from it consists of four parts

Initialization & Permission *(>folder “manifests”)*: A manifest file is in the project’s root directory. It presents essential information about the application to the android system. So the system must have before it can run any of the application's code.

Logics *(>folder “java” )*: Java files contain logics used to control the application’s activities.

UI *(>folder “res”)*: This folder keeps Xml files about the application’s UI (User Interface).

Gradle Scripts: These scripts are used to automate and manage the build process.



Genymotion (See more, Download & Set up: <https://www.genymotion.com> ) : Genymotion is a fast third-party emulator that can be used instead of the default Android Emulator / Android Smartphone (In some cases it's as good as or better than developing on actual devices!)



Android Smartphone: Android smartphone is a mobile phone with the advanced android operating system.



1.2.2 Server Side (See more, Download & Set up: <https://aws.amazon.com> ): In this project, we use amazon server.



1.2.2.-2.1 API (Php)

Slim (See more, Download & Set up: <http://www.slimframework.com> ): Slim is a Php micro framework that helps you quickly write simple yet powerful web applications and APIs. In this project, We already set up Slim. It is ready to use. You just push all files about it to your server and it will work !



Sublime Text (See more, Download & Set up: <https://www.sublimetext.com> ) : Sublime text a sophisticated text editor for code, markup and prose.



WinSCP (See more, Download & Set up: <https://winscp.net/eng/download.php> ) : WinSCP is a free and open-source SFTP, FTP, WebDAV and SCP client for Microsoft Windows. Its main function is secure file transfer between a local and a remote computer. In this project, It is used to push files from a computer to the server.



1.2.2.-2.2 Database (MySQL)

HeidiSQL (See more, Download & Set up: <http://www.heidisql.com>) : HeidiSQL a tool designed for web developers using the popular MySQL server, Microsoft SQL databases and PostgreSQL. It enables you to browse and edit data, create and edit tables, views, procedures, triggers and scheduled events. Also, you can export structure and data either to SQL file, clipboard or to other servers. In this project, It is used to mangage data in the amazon database server.



1.2.3 Repository

Github (See more: <https://github.com> ): Github is a web-based git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.



SourceTree (See more, Download & Set up: <https://www.atlassian.com/software/sourcetree>): SourceTree simplifies how you interact with your Git and Mercurial repositories so you can focus on coding. Visualize and manage your repositories through SourceTree's simple interface.



1. **Sending & Receiving Data Implementation**

**//Please use your own server (including changing url in the code to your own )**

The steps below are about the sending & receiving data implementation of “BeLeaf”. They are about the flow of sending & receiving data between the android application and the server side.

* + 1. Requirement: Library “Volley” (See more, Download & Set up: 1.<http://stackoverflow.com/questions/21065477/best-way-to-incorporate-volley-or-other-library-into-android-studio-project>

2.<https://developer.android.com/training/volley/index.html>

): In this project, we have already set up library “Volley”

These are steps of library”Volley” installation & set up in this project

2.0.2 Download “volley.rar” and put it in the folder project->app->libs

2.0.3 Put the code below in build.gradle(Module.app). So it will look like this.

dependencies {

…

compile **'com.android.volley:volley:1.0.0'**

…

}

2.0.4 Put the code about connection permissions in AndroidManifest.xml. So it will look like this.

<**manifest xmlns:android="http://schemas.android.com/apk/res/android"**

**package="com.latte.oeuff.suicidepreventionapp"**>

…

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

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

…

</**manifest**>

* 1. Request (See all codes in BeLeaf Project)

This is a first step that the application sends a data request (Json format) to API file for requesting the data.

*Example Codes in the application (in java class “SurveyHistory\_Cage”):*

**import** com.android.volley. . . . (After putting all codes in a java class and still not importing all related libraries, you can put the cursor to an error line and press “alt + enter” for auto importing )

*//These are variables essential in the process*

*//\*\*\*\*\*\*\*\*\*\*\*\* Volley \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**RequestQueue **requestQueue**;  
**static** TrustManager[] *trustManagers*;  
**static final** X509Certificate[] ***\_AcceptedIssuers*** = **new** X509Certificate[]{};

*//onCreate()*

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\* Volley \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****requestQueue** = Volley.*newRequestQueue*(**this**);

*//These are the set up code of the variable “requestQueue” which is an object that //keeps requests in a queue.*

*//------------------ seesurveyhistory\_cage --------------------------------***public void** seesurveyhistory\_cage(){  
 HttpsTrustManager.*allowAllSSL*(); *//Trusting all certificates  
 //String url = "http://ahealth.burnwork.space/vip/myapp/suicidePreventionAPIs.php/seesurveyhistory";* String url = **"http://auth.oeufhp.me/beleaf.php/seesurveyhistory\_cage"**;  
 *//---------Message----------------* **final** ProgressDialog pd = **new** ProgressDialog(SurveyHistory\_Cage.**this**);  
 pd.setMessage(**"loading..."**);  
 pd.show();  
  
*//----------POST Request---------------  
//https://github.com/codepath/android\_guides/wiki/Networking-with-the-Volley-Library* StringRequest postRequest = **new** StringRequest(Request.Method.***POST***, url, **new** Response.Listener<String>() {  
 @Override  
 **public void** onResponse(String response) {  
  
 **try** { //This part belongs to “5. Response and its usage”  
 Log.*d*(**"seesurvey cage res"**,response);  
 *//-----------My logics---------------* String seesurveyhistory=**""**;  
  
 *//1 get value(=String) from response(=json array)* JSONObject jsonResponse = **new** JSONObject(response);  
 String stringResponse = jsonResponse.getString(**"seesurveyhistory\_cage"**);  
  
 *//2 change type from String->json array  
 //http://stackoverflow.com/questions/9961018/getting-specific-value-from-jsonarray* JSONArray jsonarray\_stringResponse = **new** JSONArray(stringResponse);  
  
 **for**(**int** i=0; i < jsonarray\_stringResponse.length();i++){  
 *//3 get value(=json object = "one jsonObject" which is json array) from json array* JSONObject jsonobject\_jsonarray\_stringResponse = jsonarray\_stringResponse.getJSONObject(i);  
  
 *//4 get value(= String = each value in "one jsonObject") from "one jsonObject"* String username = jsonobject\_jsonarray\_stringResponse.getString(**"username"**);  
 *// String password = jsonobject\_jsonarray\_stringResponse.getString("password");* String sentdate = jsonobject\_jsonarray\_stringResponse.getString(**"sentdate"**);  
 String survey\_q1\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q1\_ans"**);  
 String survey\_q2\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q2\_ans"**);  
 String survey\_q3\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q3\_ans"**);  
 String survey\_q4\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q4\_ans"**);  
 String totalscore = jsonobject\_jsonarray\_stringResponse.getString(**"totalScore"**);  
  
 seesurveyhistory = **" Record:"**+(i+1)+**" "**+**"username:"**+username+**" "**+**"sentdate:"**+sentdate+**" "** +**"Q1ans:"**+survey\_q1\_ans+**" "** +**"Q2ans:"**+survey\_q2\_ans+**" "** +**"Q3ans:"**+survey\_q3\_ans+**" "** +**"Q4ans:"**+survey\_q4\_ans+**" "** +**"total score:"**+totalscore+**" "**;  
 *//-------------- Show survey history ---------------------------------------* TextView aline =**new** TextView(SurveyHistory\_Cage.**this**); *//create a textview without binding to XML file* aline.setText(seesurveyhistory);  
 **seesurveyhistory\_layout**.setBackgroundColor(Color.***TRANSPARENT***);  
 **seesurveyhistory\_layout**.addView(aline); *//add that textview in the LinearLayout* System.*getProperty*(**"line.separator"**); *//go to a new line* TextView space = **new** TextView(SurveyHistory\_Cage.**this**);  
 space.setText(**"-------------------------------------------"**);  
 **seesurveyhistory\_layout**.addView(space);  
 System.*getProperty*(**"line.separator"**);  
  
 *//----- if try is success -> dismiss the dialog ---------* pd.dismiss(); *//Dismiss & Removing it from the screen* }  
  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 },  
 **new** Response.ErrorListener() {  
 @Override  
 **public void** onErrorResponse(VolleyError error) {  
 error.printStackTrace();  
  
 *//-----------Check error (useful !)-----------------------------------------------* NetworkResponse networkResponse = error.**networkResponse**;  
 **if** (networkResponse != **null**) {  
 Log.*e*(**"Volley"**, **"Error. HTTP Status Code:"**+networkResponse.**statusCode**);  
 }  
  
 **if** (error **instanceof** TimeoutError) {  
 Log.*e*(**"Volley"**, **"TimeoutError"**);  
 }**else if**(error **instanceof** NoConnectionError){  
 Log.*e*(**"Volley"**, **"NoConnectionError"**);  
 } **else if** (error **instanceof** AuthFailureError) {  
 Log.*e*(**"Volley"**, **"AuthFailureError"**);  
 } **else if** (error **instanceof** ServerError) {  
 Log.*e*(**"Volley"**, **"ServerError"**);  
 } **else if** (error **instanceof** NetworkError) {  
 Log.*e*(**"Volley"**, **"NetworkError"**);  
 } **else if** (error **instanceof** ParseError) {  
 Log.*e*(**"Volley"**, **"ParseError"**);  
 }  
 *//--------if error -> dismiss the dialog ---------* pd.dismiss(); *//Dismiss & Removing it from the screen* }  
 }  
 )  
 {

//This part is about putting data into the requestQueue   
 @Override  
 **protected** Map<String, String> getParams() {  
 Map<String, String> params = **new** HashMap<>();  
 *// the POST parameters:* params.put(**"username"**, **username**);  
 params.put(**"password"**, **password**);  
 **return** params;  
 }  
 };  
  
 **requestQueue**.add(postRequest);  
}

* 1. API’s process (See all codes in API Beleaf.php / You can use your own server to push API file to)

The second step is that the API receives the data request from the application (Json format) and quries the data matched with conditions in the API file. Then it sends back the data to the application as a response (Json format).

<?php

#\*\*\*\*\*\*\*\*\* This is my archetype source code for APIs (Using slim micro framwork) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#====================== 1. initial Setting ==================================================

//1.1 set Req and Res +autoload

use \Psr\Http\Message\ServerRequestInterface as Request;

use \Psr\Http\Message\ResponseInterface as Response;

require 'vendor/autoload.php';

//1.2 function connect\_db

function connect\_db(){

$dbhost = "ahealth.camdtseexuim.ap-southeast-1.rds.amazonaws.com"; //$dbhost = localhost:8081/phpMyAdmin -> FAILED

$dbuser = "ahealth";

$dbpass = "jarbill";

$dbname = "suicidePrevention";

$mysql\_conn\_string = "mysql:host=$dbhost;dbname=$dbname";

$dbConnection = new PDO($mysql\_conn\_string, $dbuser, $dbpass); //Start PDO connection (PDO= slim's libraey)

$dbConnection -> exec("SET CHARACTER SET utf8"); //=excutes ("...")

$dbConnection->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION); //Set Error Reporting

return $dbConnection;

}

//1.3 call slim

$app = new \Slim\App(['settings' => ['displayErrorDetails' => true]]); //$app = new \Slim\App();

…

//3.6 seesurveyhistory\_cage

$app->post('/seesurveyhistory\_cage', function(Request $req, Response $res) {

//get info from $req

$username = $req->getParsedBody()['username'];

$password = $req->getParsedBody()['password'];

try{

$db = connect\_db(); //HERE

$sth = $db->prepare ("SELECT \*

FROM survey\_cage\_ans

WHERE (username = :username) AND (password = :password) ");

$sth->bindParam(':username', $username, PDO::PARAM\_STR); //Can use with VARCHAR

$sth->bindParam(':password', $password, PDO::PARAM\_STR);

$sth->execute();

$result = $sth->fetchall(PDO::FETCH\_OBJ);

$db = null;

}

catch (PDOException $e) {

$res->withStatus(404);

echo '{"error":{"text":'. $e->getMessage() .'}}';

}

$result= array('seesurveyhistory\_cage'=>$result);

return $res->write(json\_encode($result));

});

2.3 Database (You can use your own server to keep the data)

The third step is about tables and schemas of the data and actions related to database. According to the second step, the data in the database is queried followed the conditions in the API file.

2.4 Response and its usage (See all codes in BeLeaf Project)

The last step is that the application receives the response (Json format) from the API and then utilize the data in the response…

@Override  
 **public void** onResponse(String response) {  
  
 **try** {  
 Log.*d*(**"seesurvey cage res"**,response);  
 *//-----------My logics---------------* String seesurveyhistory=**""**;  
  
 *//1 get value(=String) from response(=json array)* JSONObject jsonResponse = **new** JSONObject(response);  
 String stringResponse = jsonResponse.getString(**"seesurveyhistory\_cage"**);  
  
 *//2 change type from String->json array  
 //http://stackoverflow.com/questions/9961018/getting-specific-value-from-jsonarray* JSONArray jsonarray\_stringResponse = **new** JSONArray(stringResponse);  
  
 **for**(**int** i=0; i < jsonarray\_stringResponse.length();i++){  
 *//3 get value(=json object = "one jsonObject" which is json array) from json array* JSONObject jsonobject\_jsonarray\_stringResponse = jsonarray\_stringResponse.getJSONObject(i);  
  
 *//4 get value(= String = each value in "one jsonObject") from "one jsonObject"* String username = jsonobject\_jsonarray\_stringResponse.getString(**"username"**);  
 *// String password = jsonobject\_jsonarray\_stringResponse.getString("password");* String sentdate = jsonobject\_jsonarray\_stringResponse.getString(**"sentdate"**);  
 String survey\_q1\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q1\_ans"**);  
 String survey\_q2\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q2\_ans"**);  
 String survey\_q3\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q3\_ans"**);  
 String survey\_q4\_ans = jsonobject\_jsonarray\_stringResponse.getString(**"survey\_cage\_q4\_ans"**);  
 String totalscore = jsonobject\_jsonarray\_stringResponse.getString(**"totalScore"**);  
  
 seesurveyhistory = **" Record:"**+(i+1)+**" "**+**"username:"**+username+**" "**+**"sentdate:"**+sentdate+**" "** +**"Q1ans:"**+survey\_q1\_ans+**" "** +**"Q2ans:"**+survey\_q2\_ans+**" "** +**"Q3ans:"**+survey\_q3\_ans+**" "** +**"Q4ans:"**+survey\_q4\_ans+**" "** +**"total score:"**+totalscore+**" "**;  
 *//-------------- Show survey history ---------------------------------------* TextView aline =**new** TextView(SurveyHistory\_Cage.**this**); *//create a textview without binding to XML file* aline.setText(seesurveyhistory);  
 **seesurveyhistory\_layout**.setBackgroundColor(Color.***TRANSPARENT***);  
 **seesurveyhistory\_layout**.addView(aline); *//add that textview in the LinearLayout* System.*getProperty*(**"line.separator"**); *//go to a new line* TextView space = **new** TextView(SurveyHistory\_Cage.**this**);  
 space.setText(**"-------------------------------------------"**);  
 **seesurveyhistory\_layout**.addView(space);  
 System.*getProperty*(**"line.separator"**);  
  
 *//----- if try is success -> dismiss the dialog ---------* pd.dismiss(); *//Dismiss & Removing it from the screen* }  
  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 },  
 **new** Response.ErrorListener() {  
 @Override  
 **public void** onErrorResponse(VolleyError error) {  
 error.printStackTrace();  
  
 *//-----------Check error (useful !)-----------------------------------------------* NetworkResponse networkResponse = error.**networkResponse**;  
 **if** (networkResponse != **null**) {  
 Log.*e*(**"Volley"**, **"Error. HTTP Status Code:"**+networkResponse.**statusCode**);  
 }  
  
 **if** (error **instanceof** TimeoutError) {  
 Log.*e*(**"Volley"**, **"TimeoutError"**);  
 }**else if**(error **instanceof** NoConnectionError){  
 Log.*e*(**"Volley"**, **"NoConnectionError"**);  
 } **else if** (error **instanceof** AuthFailureError) {  
 Log.*e*(**"Volley"**, **"AuthFailureError"**);  
 } **else if** (error **instanceof** ServerError) {  
 Log.*e*(**"Volley"**, **"ServerError"**);  
 } **else if** (error **instanceof** NetworkError) {  
 Log.*e*(**"Volley"**, **"NetworkError"**);  
 } **else if** (error **instanceof** ParseError) {  
 Log.*e*(**"Volley"**, **"ParseError"**);  
 }  
 *//--------if error -> dismiss the dialog ---------* pd.dismiss(); *//Dismiss & Removing it from the screen* }  
 }  
 )

References

<https://madhusudhanakn.wordpress.com/2011/05/19/android-connecting-to-mysql-using-php/>

<https://wikipedia.org>