**Chapter 3**

**RESEARCH METHODOLOGY**

This chapter presents the process and progression of process development and method in determining the comparative of two databases in able to come up on a concrete basis which is better in mobile environment. It includes the discussion of process development, computational model and experimental analysis which covers testing procedures.

**Research Design**

Database is a collection of information that organized to easily allow to be accessed, managed and update the data. Database itself to analyze the data. Database have evolved from hierarchical and network database, up today with SQL and NoSQL database.

This research provides to understand the core differences of two databases: MySQL and Firebase that use by the researchers in their GARDGIA application which can help to figure and explore the fundamental differences in terms of features and functionality, how to retrieve and store data in short time and better to use in mobile application.

In this study, comparison of database is the main objective to determine which database is better to use in handling real-time data in mobile application. MySQL and Firebase are both used to determine and test which database can be used in GARDGIA application.

Multiple Access with Volume Test with Speed / Response Time

Maintainability

Survey

Technical Comparison

User’s Feedback

Compatibility

Structure of Database

Data Gathering

Comparative Study

Performance Test

Evaluation and Conclusion

**Figure 2. Research Methodology Flowchart**

The figure above shows how the researchers can work successfully on their study by following the step by step procedure.

**Data Gathering**

Gathering data and information related to the comparative analysis of MySQL and Firebase in Mobile application were conducted. There were several tools use to perform this task including research of thesis and articles that are related in the study. Researchers also used internet to research more articles that can help researchers to understand the study.

**Comparative Study**

To further understand the study, researchers decided to use area of comparison which include survey and technical comparison to be able to compare two database and can distinguished which is better in mobile application.

1. **Survey**

Researchers created survey form related to the study to be able to get which of the two database are more capable to hold the data and efficient to use in mobile environment.

1. **User’s Feedback**

Survey have big roles in our study to be able to which of database between MySQL and Firebase have better capabilities in mobile environment.

1. **Technical Comparison**

To be able to conclude which of the two database are better in mobile environment. Researchers created list of technical comparison which can help to test the comparison of database and overview the two database have capabilities in mobile environment.

**A. Compatibility**

Researchers analyze and test both database on how they respond in GARDGIA application in corresponding software requirements.

**Table 1. Compatibility in Software Requirements**

|  |  |  |
| --- | --- | --- |
| **Compatibility** | **MySQL** | **Firebase** |
| Android Version | Android Jellybean up to latest | Android Jellybean up to latest |

Both database are able to run in android version with a minimum requirement of Jellybean version up to latest.

**Table 2. Compatibility in Hardware Requirements**

|  |  |  |
| --- | --- | --- |
| **Compatibility** | **MySQL** | **Firebase** |
| Processor | Minimum of 1.3 GHz | Minimum of 1.3 GHz |
| RAM | Minimum of 2GB | Minimum of 2GB |

Both database are able to run the GARDGIA application with a minimum of 1.3 GHz processor and minimum requirement of 2GB RAM of android phone.

**B. Structure of database**

To able to understand the comparative study, researchers analyze and understand the database structure of both database. To further understand how to use Firebase, Table 3 shows a comparison between the terms used in MySQL and Firebase respectively.

**Table 3 MySQL vs. Firebase terms**

|  |  |
| --- | --- |
| **MySQL** | **Firebase** |
| Database | Database |
| Table | Collection of Object |
| Primary Key | Object/ Key |
| Column | Name |
| Row | Value |

As shown in Table 3, in Firebase, some MySQL terms, such as table, column and row, get another name, namely object, name, value respectively. In other words, researchers can say that Firebase contains object, objects contain name and value.

The researchers create a table structure in MySQL and Firebase with the same structure. Below is the table structure of MySQL:

**CREATE TABLE user (**

**user\_id int(255) auto\_increment,**

**user\_name varchar(255),**

**user\_contact varchar(255),**

**user\_email varchar(255),**

**user\_pass varchar(255),**

**PRIMARY KEY (user\_id)**

**);**

In Firebase with the same structure of the table user in MySQL version

**{**

**user: {**

**user\_idpk1: {**

**user\_name: “John Doe”,**

**user\_contact: “09234567890”,**

**user\_email: johndoe@gmail.com,**

**user\_pass: “123456”**

**},**

**user\_idpk2: {**

**user\_name: “Jane Doe”,**

**user\_contact: “09230986543”,**

**user\_email: “janedoe@gmail.com”,**

**user\_pass: “654321”**

**}**

**}**

**}**

In the structure of database between MySQL and Firebase are different from one other in structure of database. Firebase is complicated to understand to learn the structure of its database than MySQL.

**C. Multiple Access with Volume Test in Speed / Response Time**

Multiple access is a technique which let multiple users to access the database in same situation and time. The researchers will test the two database to show how the database respond in multiple access in database. Volume testing is a technique which refers to test the software application with a certain amount of data to use. The researchers will test the two databases to show how the database respond from small data to huge data. The researchers will test both databases in terms of speed and response time in different situation.

**D. Maintainability**

Maintainability is a process which unexpected failure occurs, perform successful repair action with a given time. The researchers will conduct testing procedure of two database to show how the database respond in storing of data in unexpected crash or failure.

MySQL have Logs buttons for the unexpected shutdown or crash of xampp. MySQL have logs button to check and view the error logs. To avoid this user needs to either run XAMPP as an administrator, or use the proper command prompt method for shutting down MySQL.

As Kratos suggests, ibdata1 can delete, but this can leave user with a broken database as other pieces of users database are still in the /mysql/data/ folder.

Firebase has new features which is crashlytics which prioritize to fix issues with powerful, realtime crash reporting. It helps developers by realtime crash report that helps to track, prioritize, and fix stability issues that erode user’s application quality.

**Performance Test**

To highlight the advantages of using the non-relational database Firebase compared to the relational database MySQL, various operations were performed on the two databases. These operations are the four basic operations that can be performed on any database, namely:

1. Create (Insert)
2. Read (Select)
3. Update
4. Delete

In performing the test, the researchers create a table structure in MySQL and Firebase with the same structure. Below is the table structure of MySQL:

**CREATE TABLE user (**

**user\_id int(10) auto\_increment,**

**user\_name varchar(255),**

**user\_contact varchar(255),**

**user\_email varchar(255),**

**user\_barangay varchar(255),**

**user\_district varchar(255),**

**user\_is\_online varchar(1),**

**user\_pass varchar(255),**

**PRIMARY KEY (user\_id)**

**);**

**CREATE TABLE user\_loc (**

**user\_loc\_id int(10),**

**user\_loc\_lat varchar(255),**

**user\_loc\_lng varchar(255),**

**CONSTRAINT fk\_user\_loc FOREIGN KEY (user\_loc\_id)**

**REFERENCES user(user\_id)**

**);**

**CREATE TABLE dropoffsite (**

**dropoffsite\_id int(10) auto\_increment,**

**dropoffsite\_barangay varchar(255),**

**dropoffsite\_district varchar(255),**

**dropoffsite\_status varchar(255),**

**PRIMARY KEY (dropoffsite\_id)**

**);**

**CREATE TABLE dropoffsite\_loc (**

**dropoffsite\_loc\_id int(10),**

**dropoffsite\_loc\_lat varchar(255),**

**dropoffsite\_loc\_lng varchar(255),**

**CONSTRAINT fk\_dropoffsite\_loc FOREIGN KEY(dropoffsite\_loc\_id)**

**REFERENCES dropoffsite(dropoffsite\_id)**

**);**

In Firebase with the same structure of the table user in MySQL version

**{**

**user:{**

**user\_idpk1:{**

**user\_email: "user@gmail.com",**

**user\_name: "John Doe",**

**user\_contact: "09236819738",**

**user\_barangay: "West Rembo",**

**user\_district: "District 2",**

**user\_is\_online:** **"0"**

**},**

**user\_idpk2:{**

**user\_email: "user2@gmail.com",**

**user\_name: "Jane Doe"**

**user\_contact: "09158280128",**

**user\_barangay: "West Rembo",**

**user\_district: "District 2",**

**user\_is\_online:** **"0"**

**}**

**},**

**user\_loc: {**

**user\_loc\_idpk1:{**

1. **"wdw4f5g1gs",**

**l:{**

* 1. **14.566821140414755,**
  2. **121.02966338396074**

**}**

**},**

**user\_loc\_idpk2:{**

1. **"wdw4f5s2z7",**

**l:{**

* 1. **14.565269706825308,**
  2. **121.03154193609953**

**}**

**}**

**},**

**dropoffsite:{**

**dropoffsite\_idpk1:{**

**dropoffsite\_barangay: "West Rembo",**

**dropoffsite\_district: "District 2",**

**dropoffsite\_status: "Not Collected"**

**},**

**dropoffsite\_idpk2:{**

**dropoffsite\_barangay: "San Antonio",**

**dropoffsite\_district: "District 1",**

**dropoffsite\_status: "Not Collected"**

**}**

**},**

**dropoffsite\_loc: {**

**dropoffsite\_idpk1:{**

**g:"wdw4f53c5k",**

**l:{**

**0: 14.56394573897020,**

**1: 121.02794073522092**

**}**

**},**

**dropoffsite\_idpk2:{**

1. **"wdw4fdvrqm",**

**l:{**

* 1. **14.562276834245909,**
  2. **121.05485066771507**

**}**

**}**

**}**

**}**

Next, the researchers created a User class in java that was used for Firebase.

*User.class*

**public class** User {

String **user\_id**;

String **user\_name**;

String **user\_contact**;

String **user\_email**;

String **user\_pass**;

**public** User(String name, String contact, String email, String pass {

**this**.**user\_name** = name;

**this**.**user\_contact** = contact;

**this**.**user\_email** = email;

**this**.**user\_pass** = pass;

}

**public** String getUser\_id() { **return user\_id**; }

**public** String getUser\_name() { **return user\_name**; }

**public** String getUser\_contact() { **return user\_contact**; }

**public** String getUser\_email() { **return user\_email**; }

**public** String getUser\_pass() { **return user\_pass**; }

**public void** setUser\_id(String user\_id) { **this**.**user\_id** = user\_id; }

**public void** setUser\_name(String user\_name) { **this**.**user\_name** = user\_name; }

**public void** setUser\_contact(String contact) { **this**.**user\_contact** = contact; }

**public void** setUser\_email(String user\_email) { **this**.**user\_email** =

user\_email; }

**public void** setUser\_pass(String user\_pass) { **this**.**user\_pass** =

user\_pass;

To store the fetched data in objects, the researchers created a simple model class shows in the code above. It is used to hold JSON data, that is why the class is having only a constructor to initialize the fields, getters to get the values and to set the values. The variable used in the class have the same name of the field in the Firebase database.

**A. Create (Insert) Operation**

The researchers began testing with the creation of databases without any content, both in Firebase and MySQL. The structure of the two databases is similar and they have about the same number of columns/fields. The common elements of the databases are: Table/Collection of object *dropoffsite* with the columns: *dropoffsite\_id, dropoffsite\_barangay, dropoffsite\_district, dropoffsite\_status* and inthe Table/Collection of object *dropoffsite\_loc* with the columns: *dropoffsite\_loc\_id,* *dropoffsite\_loc\_lat, dropoffsite\_loc\_lng*. Data insertion began in both databaseswith dropoffsite and dropoffsite location insertion. Their ID was generated automatically by both databases.

For recording the time required to insert the elements in the database, the researchers used the Java function *Timezone and Calendar*, which recorded the time from the beginning of the script runtime and until it is complete.

The insert operation has the following MySQL syntax:

<?php

include **'DatabaseConfig.php'**;

$con = mysqli\_connect($HostName,$HostUser,$HostPass,$DatabaseName);

$barangay = $\_POST[**'barangay'**];

$district = $\_POST[**'district'**];

$lat = $\_POST[**'lat'**];

$lng = $\_POST[**'lng'**];

$Sql\_Query = **"INSERT INTO dropoffsite**

**(dropoffsite\_barangay, dropoffsite\_district, dropoffsite\_status) VALUES**

**('$barangay','$district','Not Collected')"**;

**if**(mysqli\_query($con,$Sql\_Query)){

$last\_id = mysqli\_insert\_id($con);

**if**($last\_id){

$Sql\_Query = **"INSERT INTO dropoffsite\_loc**

**(dropoffsite\_loc\_id,dropoffsite\_loc\_lat,dropoffsite\_loc\_lng) VALUES**

**($last\_id,'$lat','$lng')"**;

**if**(mysqli\_query($con,$Sql\_Query)){

echo **'Data Inserted Successfully'**;

}**else**{

echo **'Try Again'**;

}

}

}**else**{

echo **'Try Again'**;

}

mysqli\_close($con);

?>

The above PHP code shows the connection of database in the server, insert query of MySQL to the database and the get post from the mobile application. The code structure checks if the Request Method is equal to POST then it will connect to the database in the server and execute the SQL query.

*MySQL Java Code*

String **HttpUrl** = **"http://gardgia.000webhostapp.com/insert\_dropoffsite.php"**; *// Creating string request with post method.*

StringRequest **stringRequest** = **new** StringRequest(Request.Method.POST, **HttpUrl**, **new** Response.Listener<String>() {

@Override

**public void** onResponse(String ServerResponse) {String finalResponse = ServerResponse.trim(); *// Matching server responce message to our text.* **if**(finalResponse.equalsIgnoreCase(**"Data Inserted Successfully"**)){

*// Output result.*

}**else** {

*// Output result.*

}

}

},

**new** Response.ErrorListener() {

@Override

**public void** onErrorResponse(VolleyError volleyError) {*// Output error.*

}

}) {

@Override

**protected** Map<String, String> getParams() {

*// Creating Map String Params.*

Map<String, String> params = **new** HashMap<String, String>();

*// Adding All values to Params.*

params.put(**"id"**, **"1050"**);

params.put(**"barangay"**, String.*valueOf*(**spinner2**.getSelectedItem()));

params.put(**"district"**, String.*valueOf*(**spinner1**.getSelectedItem())); params.put(**"latitude"**, String.*valueOf*(latLng.**latitude**)); params.put(**"longitude"**, String.*valueOf*(latLng.**longitude**))

**return** params;

}

};

*// Creating RequestQueue.*

RequestQueue **requestQueue** = Volley.newRequestQueue(MapsActivity.**this**);

*// Adding the StringRequest object into requestQueue.*

**requestQueue**.add(**stringRequest**);

The above java code shows the created StringRequest with a POST method that will send the data of the params to the URL address and pass the value in the PHP code using Volley.

The following Firebase syntax has the same function of the above PHP and Java codes.

*Firebase Java Code*

**private** GeoFire **geoFireDOSL**;

**private** DatabaseReference **refDOSL**, **mDatabase**;

**mDatabase** = FirebaseDatabase.*getInstance*().getReference();

String dropoffsiteKey = **mDatabase**.child(**"dropoffsite"**).push().getKey(); **mDatabase**.child(**"dropoffsite"**).child(dropoffsiteKey).child(**"District"**).setValue(String.*valueOf*(**spinner1**.getSelectedItem()));

**mDatabase**.child(**"dropoffsite"**).child(dropoffsiteKey).child(**"Barangay"**).setValue

(String.*valueOf*(**spinner2**.getSelectedItem()));

**mDatabase**.child(**"dropoffsite"**).child(dropoffsiteKey).child(**"Status"**).setValue(**" Not Collected"**);

**refDOSL** = FirebaseDatabase.*getInstance*().getReference(**"dropoffsites\_location"**); **geoFireDOSL** = **new** GeoFire(**refDOSL**);

**geoFireDOSL**.setLocation(dropoffsiteKey, **new** GeoLocation(latLng.**latitude**,latLng.**longitude**), **new** GeoFire.CompletionListener() {

@Override

**public void** onComplete(String key, DatabaseError error) {

**if** (error != **null**) {

* + - *Output error.*

} **else**{

* + *Output success.*

}

}

});

The above java code shows the insert operation of Firebase in the database. User class is used to get the data inserted in the mobile application and pass as a JSON object in the Firebase database.

**B. Read (Select) Operation**

To test the performance of select (query) operations, the researchers create a program that will select all the users from the database.

The select operation has the following MySQL syntax:

*select\_users.php*

<?php

include **'DatabaseConfig.php'**;

*//connecting to database and getting the connection object*

$conn = mysqli\_connect($HostName,$HostUser,$HostPass,$DatabaseName);

*//creating a query*

$stmt = $conn->prepare(**"SELECT user\_id, user\_name, user\_contact,**

**user\_email, user\_district, user\_barangay, user\_is\_online FROM user"**);

*//executing the query*

$stmt->execute();

*//binding results to the query*

$stmt->bind\_result($user\_id, $user\_name, $user\_contact, $user\_email, $user\_district, $user\_barangay, $user\_is\_online);

$users = array();

*//traversing through all the result*

**while**($stmt->fetch()){

$temp = array();

$temp[**'user\_id'**] = $user\_id;

$temp[**'user\_name'**] = $user\_name;

$temp[**'user\_contact'**] = $user\_contact;

$temp[**'user\_email'**] = $user\_email;

$temp[**'user\_district'**] = $user\_district;

$temp[**'user\_barangay'**] = $user\_barangay;

$temp[**'user\_is\_online'**] = $user\_is\_online;

array\_push($users, $temp);

}

*//displaying the result in JSON format*

echo JSON\_encode($users);

?>

The previous PHP code shows the connection of database in the server, select query of MySQL from the database and the fetched data that will be pass to the mobile application using JSON\_encode.

*MySQL Java Code*

String **HttpUrl** = **" http://gardgia.000webhostapp.com/select\_users.php"**;

*/\**

* *Creating a String Request.*
* *The request type is GET defined by first parameter.*
* *The URL is defined in the second parameter.*
* *Then we have a Response Listener and a Error Listener.*
* *In response listener we will get the JSON response as a String.*

*\*\*/*

StringRequest **stringRequest** = **new** StringRequest(Request.Method.GET, **HttpUrl**, **new** Response.Listener<String>() {

@Override

**public void** onResponse(String response) {

**try** {

*// Converting the string to JSON array object.*

JSONArray array = **new** JSONArray(response);

*//traversing through all the object*

**for** (**int** i = 0; i < array.length(); i++) {

*// Getting user object from JSON array.* JSONObject user = array.getJSONObject(i);

*// Output user.*

}

} **catch** (JSONException e) {

e.printStackTrace();

}

}

},

**new** Response.ErrorListener() {

@Override

**public void** onErrorResponse(VolleyError error) {

}

});

*// Adding our stringrequest to queue.* Volley.newRequestQueue(**this**).add(**stringRequest**);

The above Java code shows the created StringRequest with a GET method that will get the data from the URL address. The data will be pass to the JSONObject as a JSON\_encode and display it to mobile application using Volley.

The following Firebase syntax has the same function of the previous PHP and Java codes.

*Firebase Java Code*

*// Declare Firebase Database Reference.*

DatabaseReference **mDatabase**;

**mDatabase** = FirebaseDatabase.getInstance().getReference();

*// Select all data of tbl\_user.*

**mDatabase**.child(**"user"**).addValueEventListener(**new** ValueEventListener() {

@Override

**public void** onDataChange(DataSnapshot dataSnapshot) {

**for** (DataSnapshot postSnapshot: dataSnapshot.getChildren()) { User user = postSnapshot.getValue(User.**class**);

*// Output user*

}

}

@Override

**public void** onCancelled(DatabaseError databaseError) {

}

});

The above Java code shows the select operation of Firebase to the database. User class is used to get the data in the user table as a JSON object and display the data in the mobile application.

**C. Update Operation**

To test the update operation in the databases, the following query were executed to update the barangay, district and location of the user in the database.

The update operation has the following MySQL syntax:

*update\_user\_loc.php*

<?php

**if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**){

include **'DatabaseConfig.php'**;

$con = mysqli\_connect($HostName,$HostUser,$HostPass,$DatabaseName);

$id = $\_POST[**'id'**];

$barangay = $\_POST[**'barangay'**];

$district = $\_POST[**'district'**];

$lat = $\_POST[**'latitude'**];

$lng = $\_POST[**'longitude'**];

$Sql\_Query = **"UPDATE user, user\_loc SET user\_loc\_lat = '$lat',**

**user\_loc\_lng = '$lng', user\_barangay = '$barangay', user\_district = '$district' WHERE user\_loc\_id = user\_id AND user\_id = $id"**;

**if**(mysqli\_query($con,$Sql\_Query))

{

echo **'Record Updated Successfully'**;

}

**else**

{

echo **'Something went wrong'**;

}

}

mysqli\_close($con);

?>

The above PHP code shows the connection of database in the server, update query of MySQL to the database and the get post from the mobile application. The code structure checks if the Request Method is equal to POST then it will connect to the database in the server and execute the SQL query.

*MySQL Java Code*

String **HttpUrl** = **"http://gardgia.000webhostapp.com/update\_user\_loc.php"**;

*// Creating string request with post method.*

StringRequest **stringRequest** = **new** StringRequest(Request.Method.POST, **HttpUrl**,

**new** Response.Listener<String>() {

@Override

**public void** onResponse(String ServerResponse) {String finalResponse = ServerResponse.trim();

*// Matching server responce message to our text.* **if**(finalResponse.equalsIgnoreCase(**"Record Updated Successfully"**)) {

*// Output result.*

}**else** {

*// Output result.*

}

}

},

**new** Response.ErrorListener() {

@Override

**public void** onErrorResponse(VolleyError volleyError) {*// Output error.*

}

}) {

@Override

**protected** Map<String, String> getParams() {

*// Creating Map String Params.*

Map<String, String> params = **new** HashMap<String, String>();

*// Adding All values to Params.*

params.put(**"barangay"**, String.*valueOf*(**spinner2**.getSelectedItem())); params.put(**"district"**, String.*valueOf*(**spinner1**.getSelectedItem())); params.put(**"lat"**, String.*valueOf*(latLng.**latitude**)); params.put(**"lng"**, String.*valueOf*(latLng.**longitude**));

**return** params;

}

};

*// Creating RequestQueue.*

RequestQueue **requestQueue** = Volley.newRequestQueue(UpdateActivity.**this**);

*// Adding the StringRequest object into requestQueue.*

**requestQueue**.add(**stringRequest);**

The above Java code shows the created StringRequest with a POST method that will send the data of the params to the URL address and pass the value in the PHP code using Volley.

The following Firebase syntax that has the same function of the previous PHP and Java codes.

*Firebase Java Code*

**private** GeoFire **geoFire**;

**private** DatabaseReference **ref**, **mDatabase**;

**mDatabase** = FirebaseDatabase.*getInstance*().getReference();

**ref** = FirebaseDatabase.*getInstance*().getReference(**"user\_loc"**);

**geoFire** = **new** GeoFire(**ref**);

**mDatabase**.child(**"user"**).child(**user**.getUid()).child(**"user\_district"**).setValue(String.*valueOf*(**spinner1**.getSelectedItem()));

**mDatabase**.child(**"user"**).child(**user**.getUid()).child(**"user\_barangay"**).setValue(String.*valueOf*(**spinner2**.getSelectedItem()));

**geoFire**.setLocation(**user**.getUid(), **new** GeoLocation(latLng.**latitude**,latLng.**longitude**), **new** GeoFire.CompletionListener() {

@Override

**public void** onComplete(String key, DatabaseError error) {

**if** (error != **null**) {

* + *Output error.*

}**else** {

* + *Output success.*

}

}

});

The above Java code shows the update operation of Firebase in the database. It shows the declaration of Firebase DatabaseReference and the query to update the data from database.

**D. Delete Operation**

As the other operations described above, the researchers have executed delete queries for each database. The query deletes the dropoffsite and dropoffsite location by inserting the dropoffsite id.

The delete operation has the following MySQL syntax:

*delete\_dropoffsite.php*

<?php

**if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**){

include **'DatabaseConfig.php'**;

$con = mysqli\_connect($HostName,$HostUser,$HostPass,$DatabaseName);

$ID = $\_POST[**'id'**];

$Sql\_Query = **"DELETE dropoffsite, dropoffsite\_loc FROM dropoffsite JOIN** **dropoffsite\_loc WHERE dropoffsite\_id=dropoffsite\_loc\_id AND dropoffsite\_id = '$ID'"**;

**if**(mysqli\_query($con,$Sql\_Query))

{

echo **'Record Deleted Successfully'**;

}

**else**

{

echo **'Something went wrong'**;

}

}

mysqli\_close($con);

?>

The above PHP code shows the connection of database in the server, delete query of MySQL to the database and the get post from the mobile application. The code structure checks if the Request Method is equal to POST then it will connect to the database in the server and execute the SQL query.

*MySQL Java Code*

String **HttpUrl** = **"http://gardgia.000webhostapp.com/delete\_user.php"**;

*// Creating string request with post method.*

StringRequest **stringRequest** = **new** StringRequest(Request.Method.POST, **HttpUrl**, **new** Response.Listener<String>() {

@Override

**public void** onResponse(String ServerResponse) {String finalResponse = ServerResponse.trim(); *// Matching server responce message to our text.*

**if**(finalResponse.equalsIgnoreCase(**"Record Deleted Successfully"**)){

*// Output result.*

}

**else** {

*// Output result.*

}

}

},

**new** Response.ErrorListener() {

@Override

**public void** onErrorResponse(VolleyError volleyError) {

*// Output error.*

}

}) {

@Override

**protected** Map<String, String> getParams() {

*// Creating Map String Params.*

Map<String, String> params = **new** HashMap<String, String>();

*// Adding All values to Params.*

params.put(**"id"**, id);

**return** params;

}

};

*// Creating RequestQueue.*

RequestQueue **requestQueue** = Volley.newRequestQueue(DeleteActivity.**this**);

*// Adding the StringRequest object into requestQueue.*

**requestQueue**.add(**stringRequest**);

The above Java code shows the created StringRequest with a POST method that will send the data of the params to the URL address and pass the value in the PHP code using Volley.

And the following Firebase syntax that has the same function of the previous PHP and Java codes.

*Firebase Java Code*

*// Declare Firebase Database Reference.*

DatabaseReference **mDatabase**;

**mDatabase** = FirebaseDatabase.getInstance().getReference();

*// Delete data from database.*

**mDatabase**.child(**"dropoffsites"**).child(id).removeValue();

**mDatabase**.child(**"dropoffsites\_location"**).child(id).removeValue();

The above Java code shows the delete operation of Firebase in the database. It shows the declaration of Firebase DatabaseReference and the query to delete the data from database

**Evaluation and Conclusion**

In this step process, the researchers will conduct evaluation of data gathered in testing procedure and analyzed both result to conclude which of two database is better for mobile application.

**Experimental Analysis**

**Test Setup**

To be able to make this research to become more reasonable, various testing procedure have been done by the researchers. A test environment is a setup of software and hardware to execute test cases. The test environment is a test where researchers allowed to test the application with databases. To able to determine the test environment, key area to set up includes:

**Software Requirements**

**Table 4. Software Requirements**

|  |  |
| --- | --- |
| Software | Description |
| Android Phone | Android Jelly Bean up to latest |
| PHP | PHO 7.0.27 |
| MySQL | MySQL 5.7.21 |
| Firebase | Firebase 1.1.5 |

The application was tested in Android Phone with corresponding version of Android Jellybean up to latest and PHP and XAMPP for MySQL database and Firebase was executed through Firebase official webpage.

**Hardware Requirements**

**Table 5. Hardware Requirements**

|  |  |
| --- | --- |
| **Hardware** | **Requirement** |
| RAM | Minimum of 2GB RAM |
| Processor | Minimum of 1.3 GHz |

Because the test results depend on the android phone which these tests are carried out, it is important to note that all the results presented were obtained from studies conducted on an android phone with the following specifications: Android 5.1 Jelly Bean with RAM of 2GB at least.

Aside from software and hardware requirements, network configuration is one of the important things to use in gathering data in testing procedure, especially in real-time tracking location of GARDGIA application and retrieving and updating the database. Speed of performance will be based on various Internet Service Providers such as Globe Telecom.

**Key Person and their Roles**

The following people are involved in test environment procedure:

**Developer**

The developer is responsible for providing the coverage of testing procedure and measuring data tools needed for record the data.

**Test Leader**

The Test Leader involved in planning, monitoring and controls during in overall testing activities.

**Tester**

The tester will be the one who performed the testing procedure, monitor the speed performance and record all the data based on the output during testing procedure.

**Data logs**

To be able to gather the data, the researchers need to identify the test data which particularly testing the application in both databases (MySQL and Firebase). Create, insert, update, delete are used and selected by the researchers for data use.

**Test Procedure**

Test procedure defines the process of the application through analyzing and testing the corresponding data and features of the application which can help the researchers to determine which among the databases are best suited in mobile application.

**Table 6. Test Procedure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Roles** | **Developer** | **Test Leader** | **Tester** |
| **Phases** | Provide coverage of testing procedure and measuring tool for recording the data | Planning, Monitor, controlling and Setup the testing activities | Records and Monitor performance during testing procedure |
| **Expected Output** | Set of data | Set of data for testing procedure | Records the output of testing procedure |

**Test Cases**

Test cases is a set of data which helps the researchers through a sequence of steps to evaluate the possible inputs and expected outputs for test cases.

**Table 7. Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Cases #** | **Description** | **Expected Output** | **Actual Output** |
| 1 | Create Drop Off Site and Drop Off Site Location | The result is expected to be able to create a drop off site and drop off site location in the database. | The user is able to create a drop off site and drop off site location. |
| 2 | Update User Details and User Location | The result is expected to be able to update the user details and user location. | The user details and user location updated. |
| 3 | View User Details | The result it is expected that the user details are viewable. | The user details are displayed. |
| 4 | Delete Drop Off Site and Drop Off Site Location | Upon the deletion of drop off site and drop off site location it is expected that the details and location is removed. | The drop off site and drop off site location is removed from the database. |