

An Android App For Food Allergy Sufferers To Search For And Review Restaurants

DT265

Higher Diploma in Computing

**Name: Maeve Rooney**

**Supervisor: Michael Collins**

**<Second Reader>**

School of Computing

Dublin Institute of Technology

**<Date>**



Abstract

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<Student Name>

<Date>

Acknowledgements

Body text

Table of Contents

1 Introduction 8

1.1 Background 8

1.2 Project Objectives 8

1.3 Project Challenges 8

2 Technologies Researched 8

2.1 Introduction 8

2.2 Eclipse and Java 9

2.3 RESTful Services andAndroid 11

2.4 Third party web server – x10hosting.com 13

2.5 FTP Program – CyberDuck 14

2.6 TextWrangler and php, mysql 15

2.7 Android platform 17

2.8 Google Maps API 19

3 Architecture & Design 21

3.1 Introduction 21

3.2 System Architecture 22

3.3 Use Case Design 25

3.4 Design Methodology 26

3.5 Development Environment 26

3.6 Web Server and MySQL Design 27

3.7 Local SQLite Design 30

3.8 User Interface Design 31

3.9 List of Features 33

4 Development & Implementation 33

4.1 Details of each component within the project, problems encountered and resolved, challenges overcome or worked around. 33

4.2 Identify key development components 33

5 System Validation 34

5.1 Testing 34

5.2 Testing Methodologies 34

5.3 Scenarios Tested 34

5.4 Functional Testing 35

5.4.1 Robotium 35

5.5 Performance and Stress Testing of Web Server 35

5.5.1 Pylot – Web Performance Tool 35

6 Project Plan 36

6.1 Project Plan analysis and review of how it changed from the initial proposal including explanation of what changed and why, and suggestions on how to address this if the project was repeated. 36

7 Conclusion 36

7.1 Analysis of the projects key elements identify the key learning obtained from the project and recommendations and suggestions for how the work can be improved on continued into the future. 36

8 Bibliography 37

8.1 Works Cited 37

8.2 Images 37

8.3 Image References 38

9 Appendix 38

­

*(Note: For bibliography use IEEE or Harvard referencing styles)*

**Problems**

* Starting activity from map overlay item- finding context
* Passing paramters to async task
* Using name pair values to post http request to php
* Getting context in itemized overlay class
* Pass restaurant id into new activity to change content dynamically
* Progress dialog while switching activities
* Phone and email
* Mergeadapter for listview with header in restaurantview
* Store passwords with salt encryption
* Use ajax to check if unique username or email already in use
* Favoutrite icons on map markers
* Check if user logged in on map and list view
* Show only restaurants with rating of more than 2 for each of users allergies
* Write review. Locate on map
* Add review to restauratn on map
* Place marker at users current location. Drag to change and make note of new geopoint
* Add heart to list view
* Blacklist
* Put rating bar indicator in alertbox

# Introduction

## Background

## Project Objectives

## Project Challenges

# Technologies Researched

## Introduction

## Eclipse and Java

The Integrated Development Envirnoment (IDE) I chose for this project is Eclipse. Eclipse is available for both Windows and Mac (my system). It is recommended on the Android Development Website. After installing Eclipse I downloaded the Android Developer Tools Plugin. This plugin comes with the latest Android platform (in this case Android 4.2 a.k.a JellyBean) and and Android system image for an emulator. While the emulator is very slow to start up the Eclipse IDE is otherwise very easy to develop Android apps in.

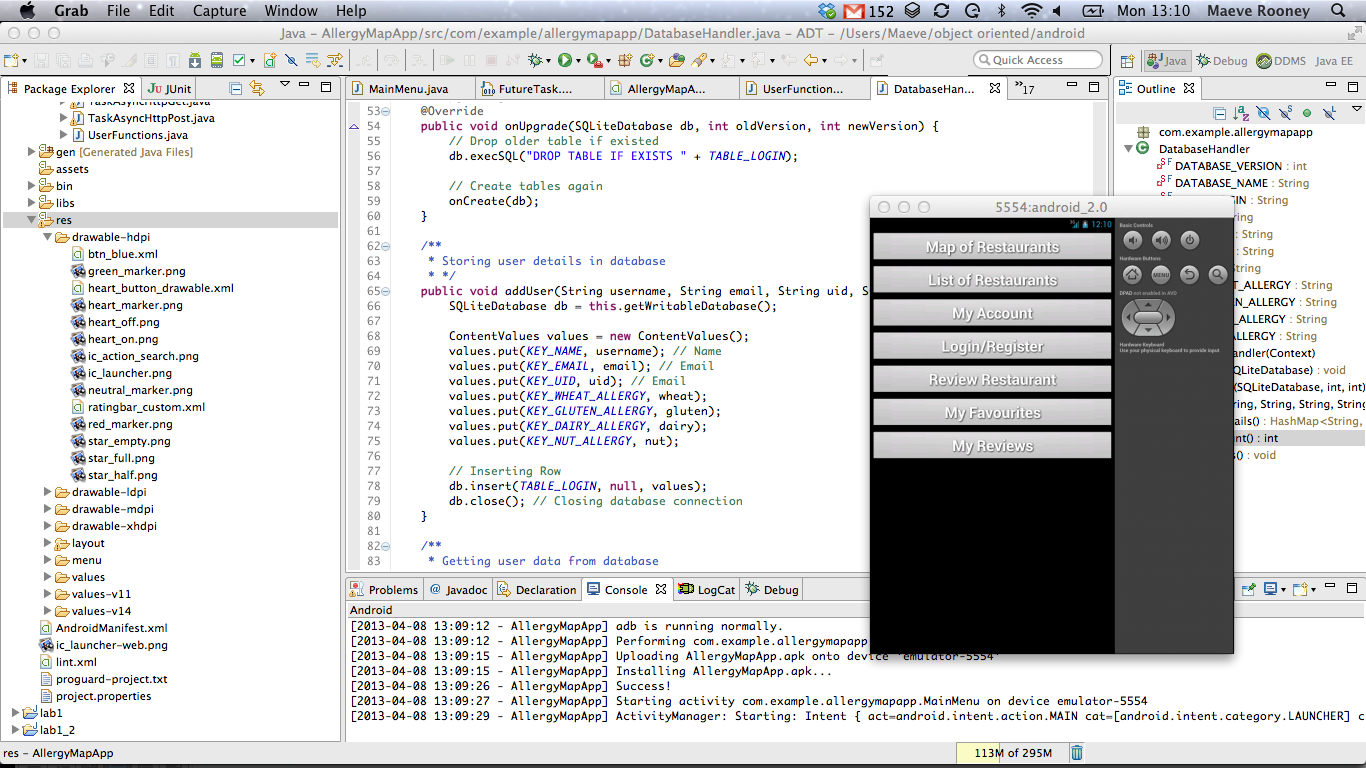


Figure 2‑1 - Eclipse for Mac with Android Emulator

Some Advantages of Developing in Eclipse

* Official Android Plugin
* Easy debugging with breakpoints and variable watch
* Spell check
* Code completion
* Code errors are highlighted
* Easy navigation between app files
* Graphical and xml view of layouts
* Easy to build and run app

Development Device

While Eclipse has an emulator to run apps on I found it to be lacking in two main areas. It is very slow to start up (up to 30 minutes) which can be frustrating when needing to test small incremental changes. No production can happen during this down time as the last increment needs to be checked before making further changes. Also as this is a map based app that utilizes the devices GPS the emulator could not effectively test this feature. It is possible to push a GPS co-ordinate to the emulator but it is not dynamic and not accurate to the users location.

For these reasons I found it necessary to test the app on a physical android device. The Samsung Galaxy Ace is Android version 2.3 (GingerBread) device. It has 2G and 3G network capabilities, a WiFi connection and an ARM 11 800 MHz CPU. The screen size is 320 x 480 pixels and 3.5 inches which is on the smaller end of Android

screen sizes. As the app is optimized for Android version 4.2, I expected there to be some difficulty running on a lower version but there have been no problems. The advantage of running on a physical device is that the GPS behaviour in the Map View and the multi-touch responses can be fully tested.

****

Figure 2‑2 - Samsung Galaxy Ace

## RESTful Services andAndroid

The RESTful (Representational State Transfer) architecture is ideal for the purposes of creating an Android App with dynamic content. It allows loose coupling between different services, in this case an Android app and a website with a MySQL database. REST does not require XML parsing [[1](#All08)]. Alternatively JSON (JavaScript Object Notation) can be used which is ideal as it is easier to parse in Java. JSON is more lightweight than XML. It uses brackets instead of tags and does not require a DTD (Document Type Definition) declaration.

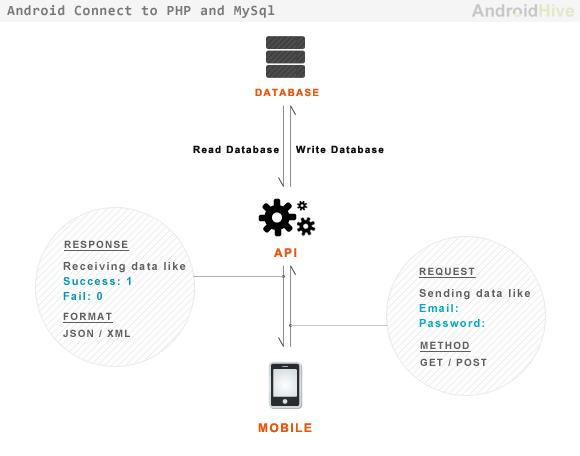
****

Figure 2‑3 - Android RESTful api

The Allergy Map App RESTful web API is implemented using the following aspects:

* The base URI for the API is http://maeverooney.x10.mx/\*
* The [Internet media type](https://en.wikipedia.org/wiki/Internet_media_type) used is JSON.
* The HTTP Methods used are GET and POST.
* The API is driven by Hypertext.
* It is Stateless. No client context stored on server between requests.
* The server and the android app are not dependant on each other. Separation of concerns.

## Third party web server – x10hosting.com

In order to run a RESTful service on the app it is necessary to setup a website with a SQL database. Many services offer free hosting. X10hosting.com offers such a service which includes MySQL databases and PHP hosting. Using these services it is easy to set up a simple website that can be used to store the content for my dynamic Android App. The content can then be retrieved and update using http calls to this website from the android device.

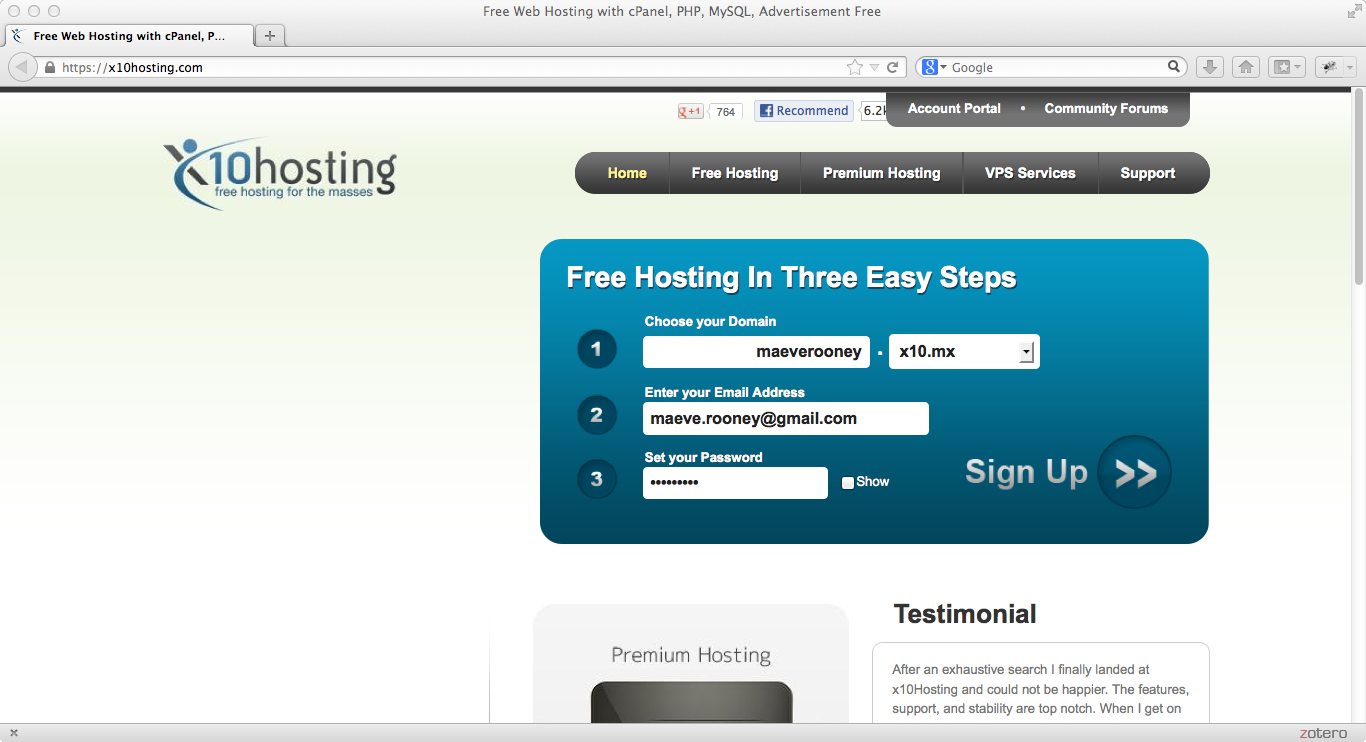


Figure 2‑4 - Signup page for hosting on x10hosting.com

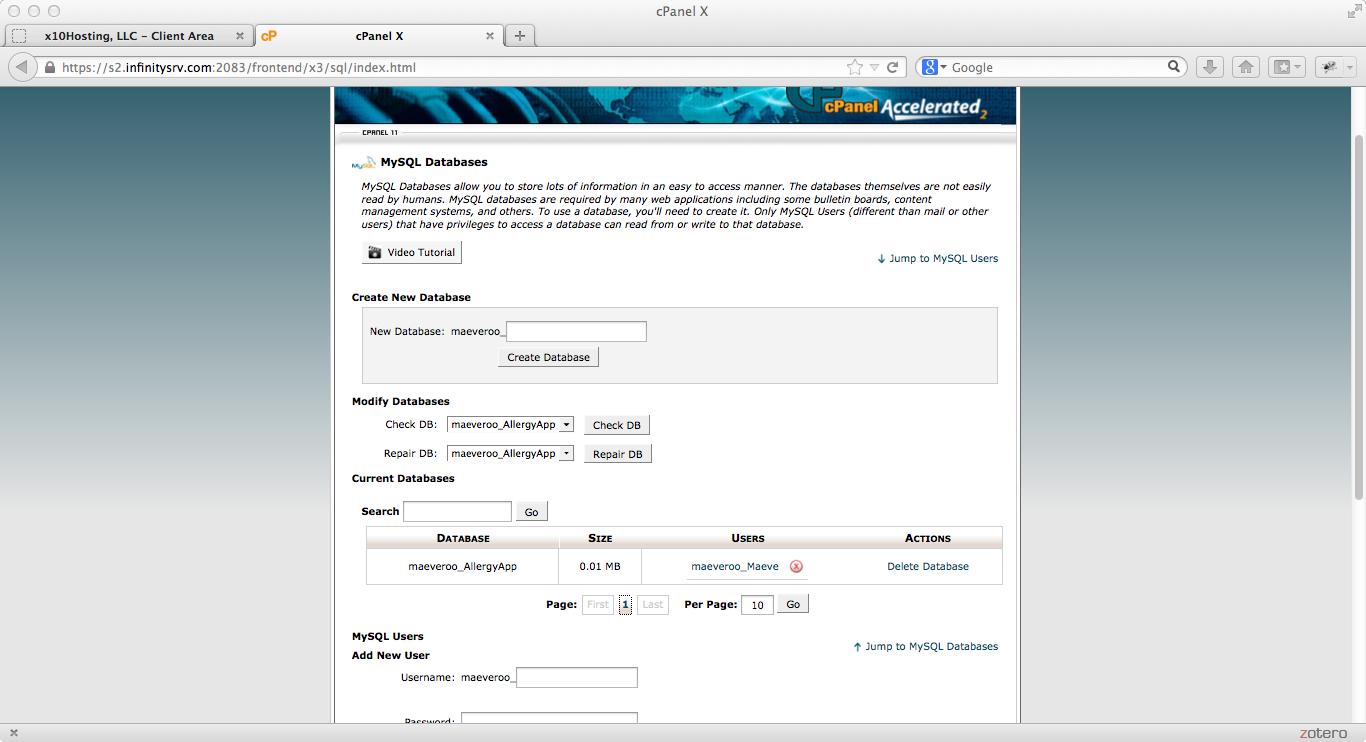
****

Figure 2‑5 - cPanel page to manage MySQL databases

## FTP Program – CyberDuck



Figure 2‑6 - CyberDuck manages file transfer

## TextWrangler and php, mysql

TextWrangler used to write php and html for web server

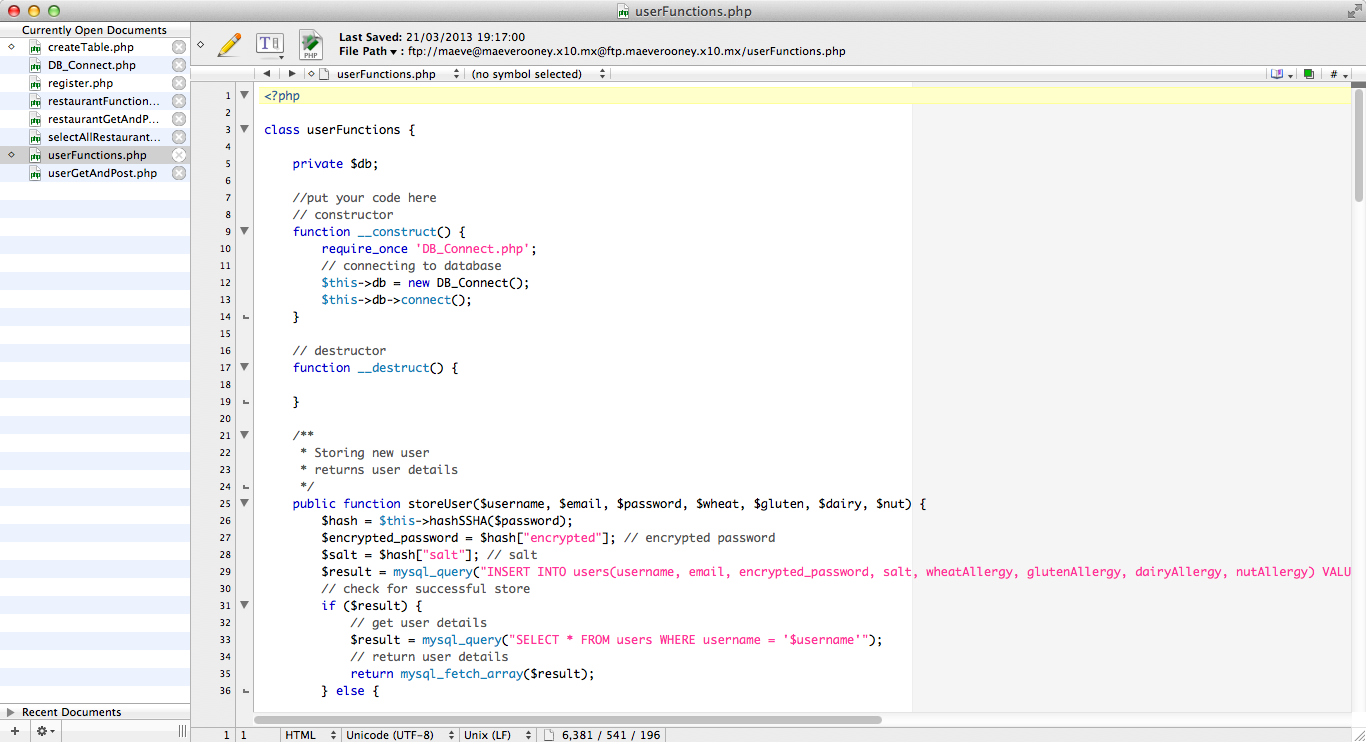


Figure 2‑7 - Textwrangler text editor

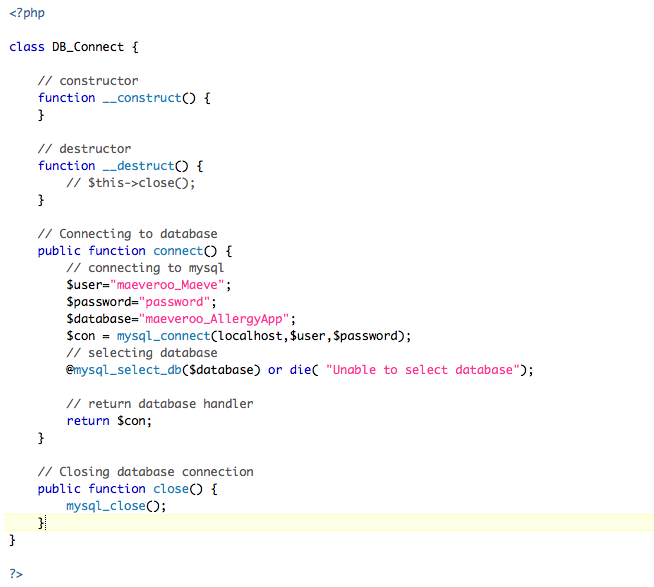
**PHP script to connect to MySQL database**

Figure 2‑8 - PHP script to connect to Database

## Android platform

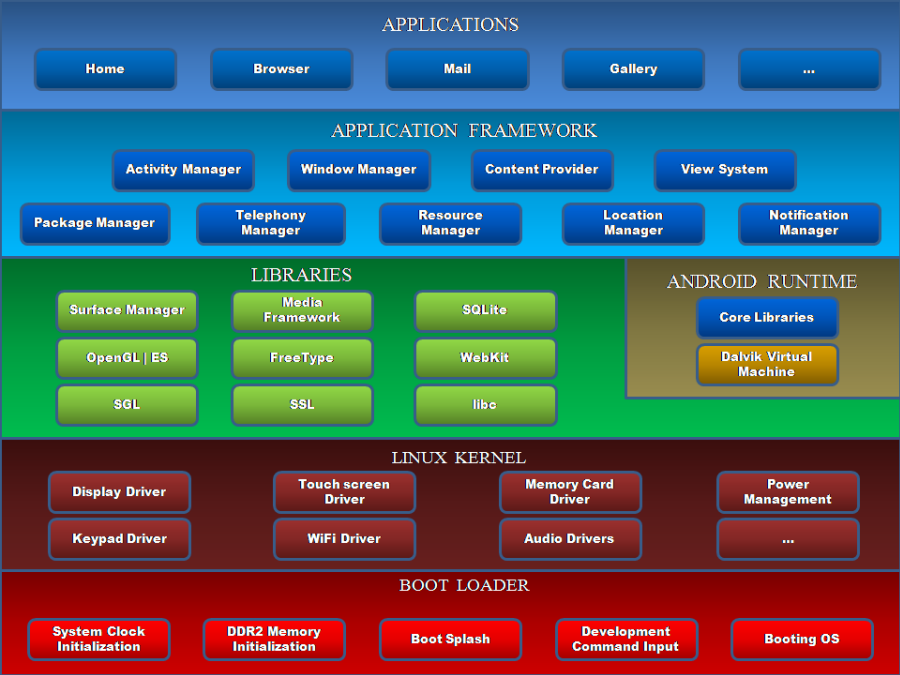


Figure 2‑9 - Android Architecture Diagram

**Versions of Android in Market 2013**

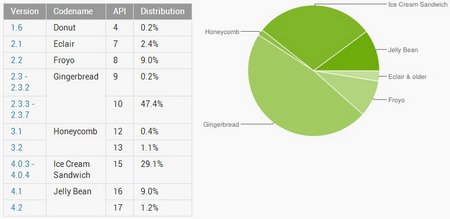
****

Figure 2‑10 - Market Share of Android Versions 2013

This app is optimised for the JellyBean (API 17) version but is working well on the Gingerbread version which the Samsung Galaxy Ace runs on.

## Google Maps API

To develop an Android app that will display Google Maps data using the API provided in the Maps external library, you must register with the service and get a Google Maps Android API v1 Key.

**Get a Google maps key for android**

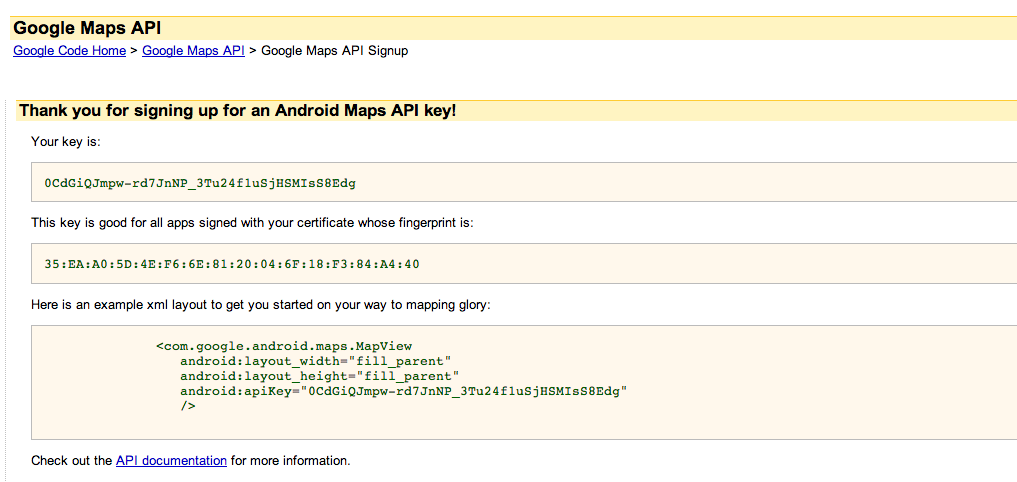
****

Figure 2‑11 - Getting Google Maps API key for Android

**Use key in xml layout of MapView activities**



Figure 2‑12 - Inserting Google API key into map layout xml

# Architecture & Design

## Introduction

## System Architecture

**Initial System Architecture Diagram**

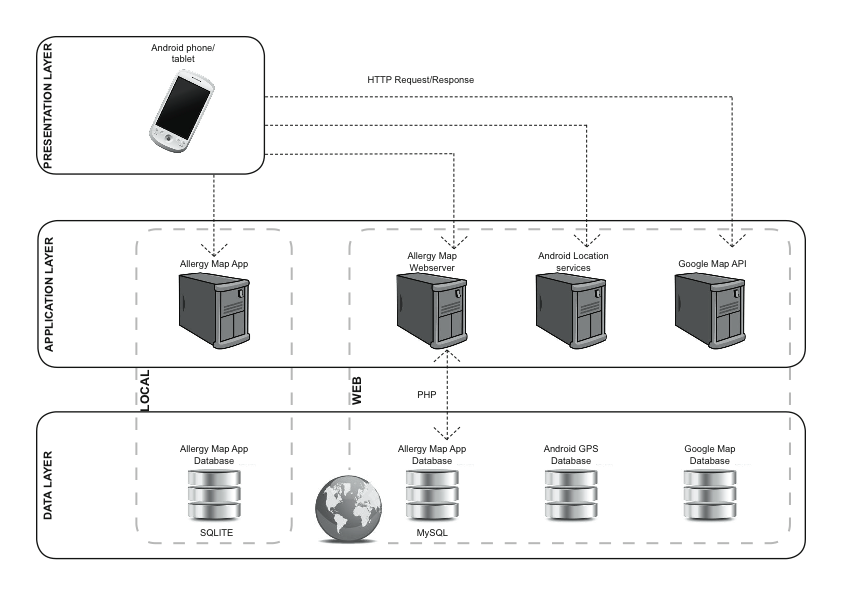
****

Figure 3‑1 - Intial System Architecture Design

**Final System Architecture Diagram**

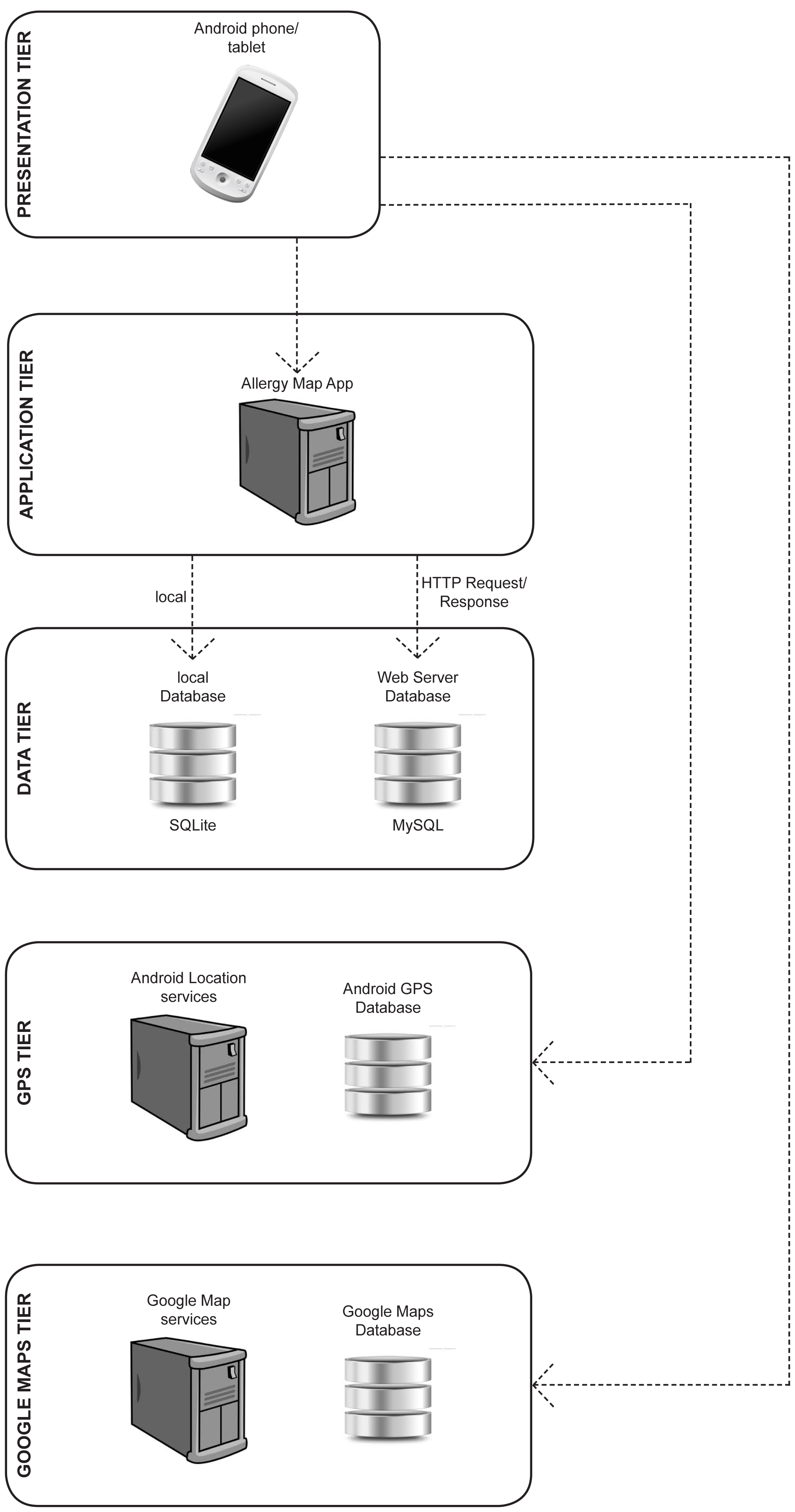


Figure 3‑2 - Final System Architecture Design

**5 Tiers**

1. Presentation Tier.
2. Application Tier.
3. Data Tier.
4. Android GPS Tier.
5. Google Maps Tier.

## Use Case Design

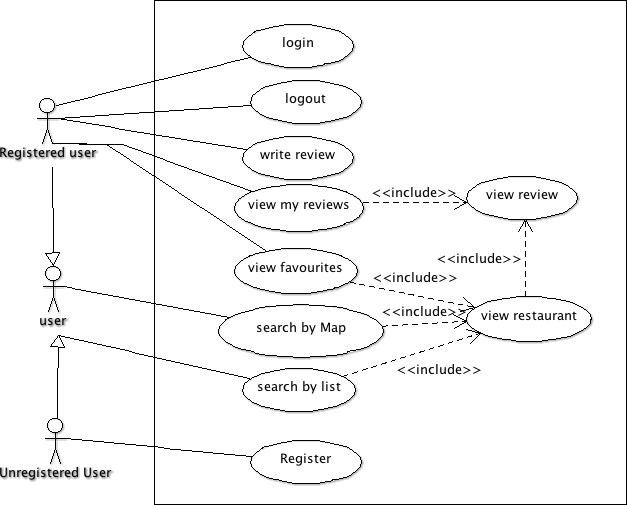
****

Figure 3‑3 - Use Case for user

## Design Methodology

## Development Environment

## Web Server and MySQL Design

**ERD Diagram for Web Server MySQL database**

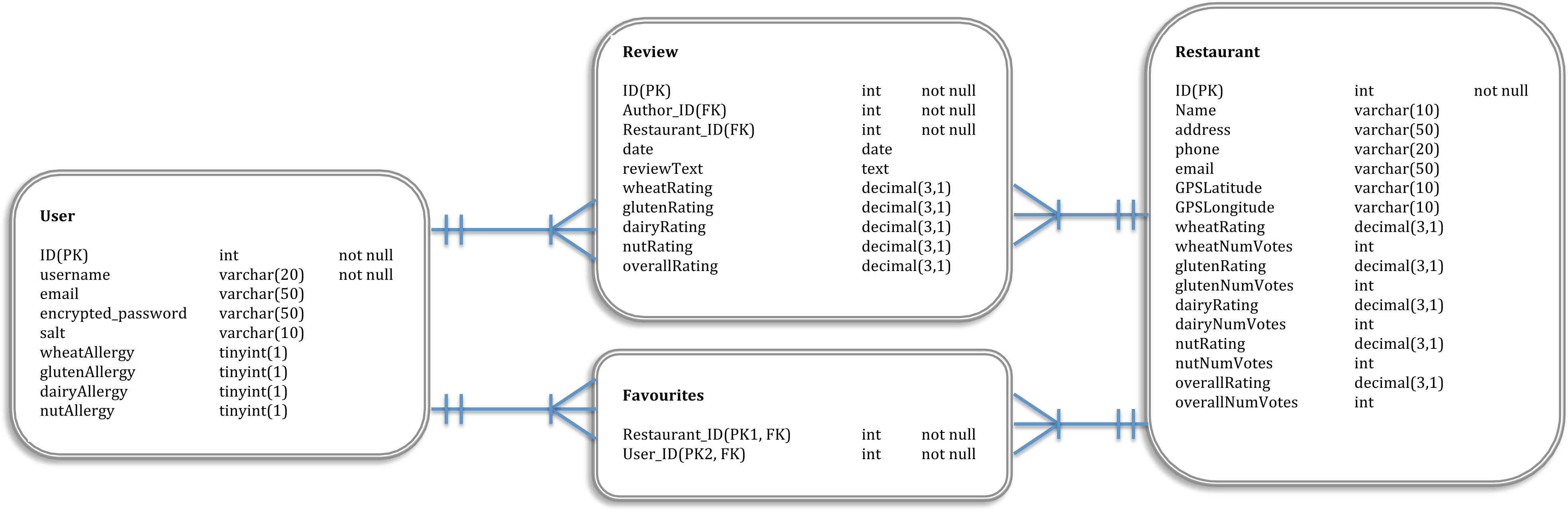
****

Figure 3‑4 - Entity Relationship Diagram for database

**User Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIELD | TYPE | SIZE(byte) | NULL | DEFAULT |
| ID (PK) | int | 4 | no | Auto-increment |
| username | varchar | 20 | No |  |
| email | varchar | 50 | no |  |
| Encrypted\_password | varchar | 50 | no |  |
| Salt | varchar | 10 | no |  |
| wheatAllergy | tinyint | 1 | yes |  |
| glutenAllergy | tinyint | 1 | yes |  |
| dairyAllergy | tinyint | 1 | yes |  |
| nutAllergy | tinyint | 1 | yes |  |

**Restaurant Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIELD | TYPE | SIZE(byte) | NULL | DEFAULT |
| ID (PK) | int | 4 | no | Auto-increment |
| name | varchar | 30 | No |  |
| address | text | <=65535 | yes |  |
| email | varchar | 50 | yes |  |
| GPSLatitude | varchar | 10 | no |  |
| GPSLongitude | varchar | 10 | no |  |
| wheatAllergy | decimal | 8 | yes |  |
| wheatNumVotes | int | 4 | yes |  |
| glutenAllergy | decimal | 8 | yes |  |
| glutenNumVotes | int | 4 | yes |  |
| dairyAllergy | decimal | 8 | yes |  |
| dairyNumVotes | int | 4 | yes |  |
| nutAllergy | decimal | 8 | yes |  |
| nutNumVotes | int | 4 | yes |  |
| overallRating | decimal | 8 | yes |  |
| overallNumVotes | int | 4 | yes |  |
| numFavourites | int | 4 | yes |  |

**Review Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIELD | TYPE | SIZE(byte) | NULL | DEFAULT |
| ID (PK) | int | 4 | no | Auto-increment |
| Author\_ID (FK) | int | 4 | no |  |
| Restaurant\_ID (FK) | int | 4 | no |  |
| date | date | 3 | no |  |
| reviewText | text | <= 65535 | yes |  |
| wheatRating | decimal | 8 | yes |  |
| glutenRating | decimal | 8 | yes |  |
| dairyRating | decimal | 8 | yes |  |
| nutRating | decimal | 8 | yes |  |
| overallRating | decimal | 8 | yes |  |

**Favourites Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIELD | TYPE | SIZE(byte) | NULL | DEFAULT |
| User\_ID (PK1, FK) | int | 4 | no |  |
| Restaurant\_ID (PK2, FK) | int | 4 | no |  |

## Local SQLite Design

**User Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIELD | TYPE | SIZE(byte) | NULL | DEFAULT |
| ID (PK) | int | 4 | no | Auto-increment |
| Unique\_ID | int | 4 | no |  |
| username | varchar | 20 | No |  |
| email | varchar | 50 | no |  |
| wheatAllergy | tinyint | 1 | yes |  |
| glutenAllergy | tinyint | 1 | yes |  |
| dairyAllergy | tinyint | 1 | yes |  |
| nutAllergy | tinyint | 1 | yes |  |

The local database stores one table that is used to check if a user is registered/logged in. It does not store the user’s password locally for security reasons. A connection to the web server database has to be made to validate the user against the password stored there. The primary key of the user in the web server database is stored in the local database in the field “Unique\_ID”. This way, any altering of the web server database (eg adding a review) can point to the correct user using the “Unique\_ID” stored locally. It minimizes the amount of calls to the web server needed, until the user logs out.

## User Interface Design

|  |  |
| --- | --- |
| Macintosh HD:Users:Maeve:Dropbox:Android Allergy App:infographics:screen grabs:main menu.tiff  Figure 0‑1 – Main Menu | Macintosh HD:Users:Maeve:Dropbox:Android Allergy App:infographics:screen grabs:register.tiff  Figure 0‑2 - Register Account |
| Macintosh HD:Users:Maeve:Dropbox:Android Allergy App:infographics:screen grabs:map view.tiff  Figure 0‑3 - Map View | Macintosh HD:Users:Maeve:Dropbox:Android Allergy App:infographics:screen grabs:list view.tiff  Figure 0‑4 - List View of restaurants |
| Macintosh HD:Users:Maeve:Dropbox:Android Allergy App:infographics:screen grabs:restaurant view.tiff  Figure 0‑5 - Restaurant Detail | Macintosh HD:Users:Maeve:Dropbox:Android Allergy App:infographics:screen grabs:review screen.tiff  Figure 0‑6 - Review Restaurant |

## List of Features

# Development & Implementation

## Details of each component within the project, problems encountered and resolved, challenges overcome or worked around.

## Identify key development components

# System Validation

## Testing

What testing was performed, why it was selected and what are the key use cases within the project.

## Testing Methodologies

### Black Box Testing

Black box testing is a good methodology to test hits app to thoroughly test the interface. It does not know see the code, it only interacts with the interface. Tests are written to see if the code behaves as expected, not to see how the code works. Black box tests are usually functional. The tests can be used to find system errors but may not be helpful in finding the case of the error. Testcases should be written to test invalid inputs from the user to see how the system handles incorrect use.

## Scenarios Tested

|  |  |
| --- | --- |
| Scenario | Classes required |
| Login with incorrect username/password | LoginActivity,  DatabaseHandler,  TaskAsyncHttpPost,  UserFunctions |
| Login with correct username/password | LoginActivity,  DatabaseHandler,  TaskAsyncHttpPost,  UserFunctions,  MainMenu |
| MainMenu navigate to RestaurantMap | MainMenu,  RestaurantMap |
| MainMenu navigate to ListRestaurants | MainMenu,  ListRestaurants |
| MainMenu navigate to MyAccount | MainMenu,  MyAccount |
| MainMenu navigate to LoginActivity | MainMenu,  LoginActivity |
| MainMenu navigate to RegisterActivity | MainMenu,  RegisterActivity |
| MainMenu navigate to ReviewRestaurant | MainMenu,  ReviewRestaurant |
| MainMenu navigate to FavouriteRestaurants | MainMenu,  FavouriteRestaurants |
| MainMenu navigate to MyReviews | MainMenu,  MyReviews |
|  |  |
|  |  |
|  |  |
|  |  |

## Functional Testing

Functional testing is used to test the activities of the android app. In particular the expected behaviours when the user interacts with the activities. Functional test are written from a users perspective. They are black box tests. They show whether or not the system is functioning as the user expects it to. To implement function test cases the jUnit class should extend the class ActivityInstrumentationTestCase2<Activity> [[2](#And13)].

### Robotium

Robotium is a third party library that extends the android test framework [[3](#Rob13)]. It provides some additional functions that make it easier to write black-box test case for Android Applications. The main Robotium class for testing is ‘solo’. Solo is initiated in the setup of the test case and used to make calls to the Activity interface. For Example:

* solo.clickOnButton(“save”);

This searches the activity for the first instance of a button with the text “save” and clicks it.

* Assert.assertTrue(solo.searchText("password"));

This searches the activity for the text “password” and validates to true if it is found.

Robotium is used to test Activites in test cases that extend the junit class ActivityInstrumentationTestCase2<Activity>.

To use Robotium the robotium.jar must be downloaded from the Robotium Website and added to the build path of the test project.

## Performance and Stress Testing of Web Server

### Pylot – Web Performance Tool

Pylot, a Web Performance tool is a free open source tool for testing performance and scalability of web services. It runs HTTP load tests, which are useful for capacity planning, benchmarking, analysis, and system [[4](#Cor13)]. Test cases can be executed from the shell or by using a GUI. The application is python based and the testcases are written using XML.

The following XML code is examples of case using get and post methods.



Figure 5‑1 - Testcase of GET method



Figure 5‑2 - Testcase of POST method

When running the testcases you select 4 factors

* The number of agents (users) making the requests
* The duration of the test in seconds
* The rampup. Which is the time span over which the agents are added in seconds. They are evenly added over this time.
* The Interval at which each user sends requests (in milliseconds)

The following table shows the results of one minute tests with a variety of number of users. The number of users making calls for the duration is static with no rampup.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Num Users | Duration  (mins) | Requests | Errors | Avg. response time  (secs) | Avg. throughput  (requests/sec) |
| 1 | 1 | 233 | 0 | 0.257 | 3.883 |
| 10 | 1 | 582 | 0 | 1.026 | 9.68 |
| 20 | 1 | 580 | 0 | 1.273 | 9.65 |
| 50 | 1 | 592 | 0 | 3.199 | 9.85 |
| 100 | 1 | 590 | 0 | 4.858 | 9.81 |
| 100 | 10 | 5415 | 108 | 9.117 | 9.02 |
| 150 | 1 | 610 | 0 | 2.755 | 10.14 |
| 150 | 10 | 6077 | 126 | 12.141 | 10.13 |

Table 1 - Sample of results from performance tests

When there is only one user the response time and throughput rates are very good. As more users are added the response time slows relative to the number of users. The number of requests per minute and the average throughput seem to plateau regardless of the number of multiple users. When the duration is increased to 10 minutes some timeout errors occur. To increase the performance a paid web service would be required.

### Sample Stress Test

* Number of users: start with 1, build to 150
* Duration of Test: 8 minutes
* Ramp Up: 8 minutes. One new user added ever 3.2 secs

**Results:**

|  |  |
| --- | --- |
| Requests | 4899 |
| Errors | 65 (timed out) |
| Data received (bytes) | 5481354 |
| Average Response Time | 7.605 secs |
| Min response time | 0.225 secs |
| Max response time | 300.139 secs |

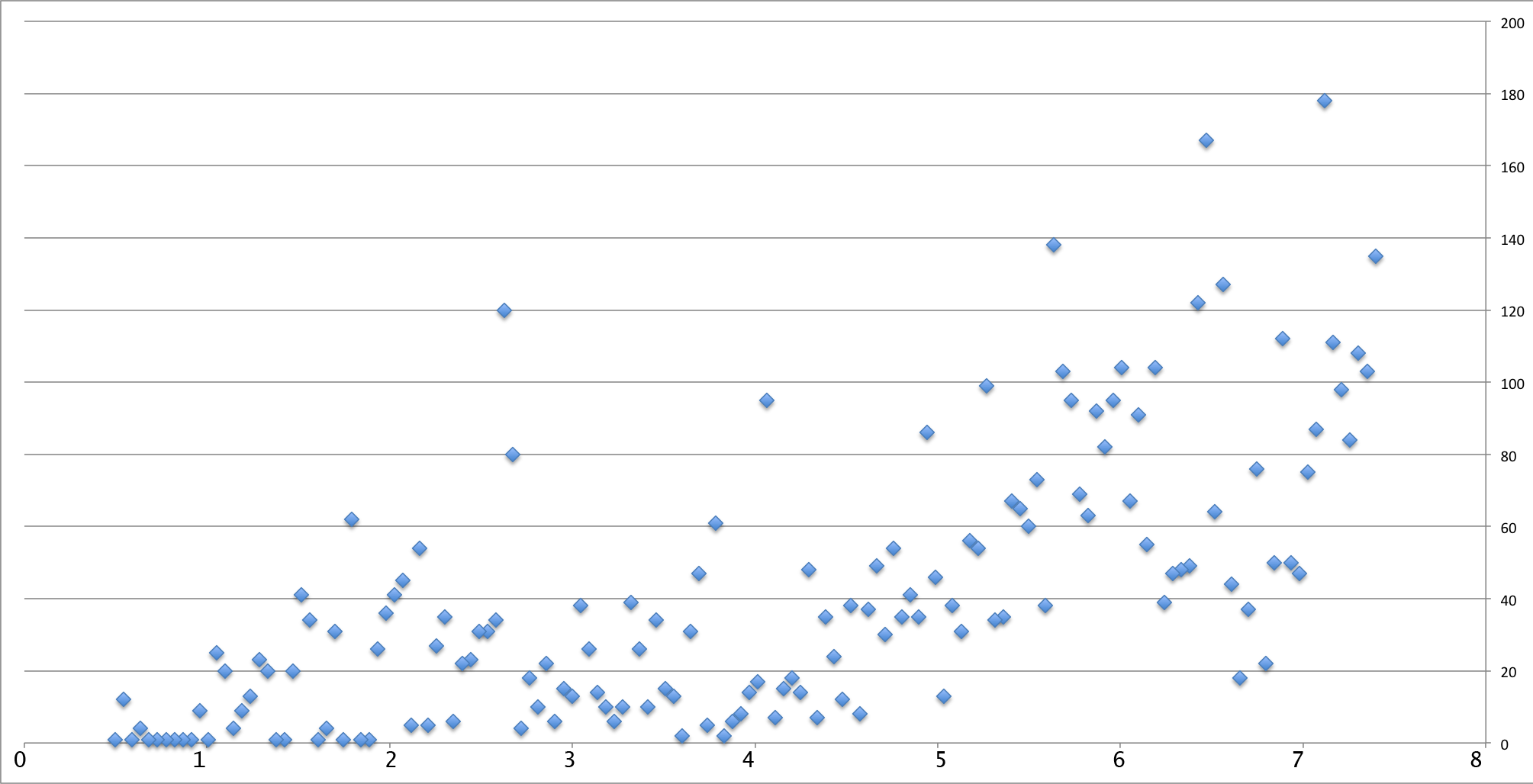


Figure 5‑3 - Number of requests per minute

### Conclusion of Web Performance Testing

The Web Server performs very efficiently with one or 2 users. As more users are added the performance slows down and plateaus to being able to handle approximately 590 requests per minute. Errors start to appear when more users are added and the test duration increased. Also shorter tests with 150 users produce no error but longer tests with the same number of users produce time out errors. The optimistic user base for this app is still relatively low therefore I don’t anticipated this being a problem.

# Project Plan

## Project Plan analysis and review of how it changed from the initial proposal including explanation of what changed and why, and suggestions on how to address this if the project was repeated.

# Conclusion

## Analysis of the projects key elements identify the key learning obtained from the project and recommendations and suggestions for how the work can be improved on continued into the future.

# Bibliography

## Works Cited

x

|  |  |
| --- | --- |
| [1] | Subbu Allamaraju. (2008, December) InfoQ. [Online]. <http://www.infoq.com/articles/subbu-allamaraju-rest> |
| [2] | Android. (2013, April) Android Developers. [Online]. <http://developer.android.com/reference/android/test/ActivityInstrumentationTestCase2.html> |
| [3] | Robotium. (2013, April) Robotium. [Online]. <https://code.google.com/p/robotium/wiki/Getting_Started> |

x

## 

## Images

Figure 2‑1 - Eclipse for Mac with Android Emulator 9

Figure 2‑2 - Samsung Galaxy Ace 9

Figure 2‑3 - Android RESTful api 10

Figure 2‑4 - Signup page for hosting on x10hosting.com 12

Figure 2‑5 - cPanel page to manage MySQL databases 12

Figure 2‑6 - CyberDuck manages file transfer 14

Figure 2‑7 - Textwrangler text editor 15

Figure 2‑8 - PHP script to connect to Database 16

Figure 2‑9 - Android Architecture Diagram 17

Figure 2‑10 - Market Share of Android Versions 2013 17

Figure 2‑11 - Getting Google Maps API key for Android 19

Figure 2‑12 - Inserting Google API key into map layout xml 19

Figure 3‑1 - Intial System Architecture Design 22

Figure 3‑2 - Final System Architecture Design 23

Figure 3‑3 - Use Case for user 25

Figure 3‑4 - Entity Relationship Diagram for database 27

Figure 3‑5 – Main Menu 32

Figure 3‑6 - Register Account 32

Figure 3‑7 - Map View 32

Figure 3‑8 - List View of restaurants 32

Figure 3‑9 - Restaurant Detail 33

Figure 3‑10 - Review Restaurant 33

Subbu Allmaraju, “[Describing RESTful Applications](http://www.infoq.com/articles/subbu-allamaraju-rest;jsessionid=6330F777CF940F156A374982DE2E73D6)” [Online] Available: <http://www.infoq.com/articles/subbu-allamaraju-rest> [Acessesd: 10-Aprl-2013]

# 

## Image References

Figure 2‑2 - Samsung Galaxy Ace

Retrieved March 19, 2013, from GSM Arena website: <http://www.gsmarena.com/samsung_galaxy_ace_s5830-pictures-3724.php>

Figure 2‑3 – Android RESTful api

Retrieved April 15, 2013, from Android Hive website: http://www.androidhive.info/2012/01/android-login-and-registration-with-php-mysql-and-sqlite/

Figure 2‑8 - Android Architecture Diagram

Retrieved March 19, 2013, from Android Developer website: <http://developer.android.com/about/versions/index.html>

Figure 2‑9 - Market Share of Android Versions 2013

Retrieved March 19, 2013, from GSM Arena website: <http://www.gsmarena.com/state_of_android_jan_2013_jb_market_share_still_under_15-news-5489.php>

Figure 2‑10 - Getting Google Maps API key for Android

Retrieved March 19, 2013, from Google Developer website:

https://developers.google.com/maps/documentation/android/index

# Appendix