DUBLIN INSTITUTE of TECHNOLOGY

Institiúid Teicneolaíochta Bhaile Átha Cliath

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

DT265

Higher Diploma in Computing

Name: Maeve Rooney

Student Number: C02008106

Supervisor: Michael Collins

Abstract

This project is an android app aimed at the niche market of food allergy sufferers. Its main purpose is to assist the user in finding a place to eat that caters to their allergy. The content of the app will be user based. The user chooses a place to dine at based on the reviews and ratings of their peers. As the content of the app is dynamic the system will be RESTful based. The content of the app will be stored on a web database and retrieved as the user interacts with the app. The aim of the project is to create a simple, intuitive app that allows for easy searching, reviewing and adding of restaurants.

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:

_____

Maeve Rooney

27/05/13

Acknowledgements

Thank you to Michael Collins for being a fantastic project Supervisor. He lent me a great guiding hand, keeping me on track and always pointing me in the right direction.

I also owe appreciation to Ciaran O'Leary and all the staff at DIT who have been very accommodation to me during a difficult personal time.

Table of Contents

Table of Tables

1 Introduction

1.1 Background

With both food allergy sufferers and smart phone users on the rise the market is primed for an app to solve an everyday problem for these niche users. The problem is where to eat. As someone with wheat intolerance it can be very difficult to find an accommodating restaurant, deli or café when looking for somewhere to eat. The same goes for other food allergy sufferers. Other than word of mouth and trial and error there are currently few solutions to this problem.

While there are some web resources out there which provide recommendations and reviews of places to eat the have their downsides [1]. Namely, they are based on the opinion of one person, they usually are aimed at only one type of allergy sufferer (e.g. celiac or lactose intolerant) and they require a lot of time to find a place that suits your location, allergy and taste.

1.2 Project Objectives

The overall aim of this project is to create an android app that provides a resource for food allergy sufferers to locate restaurants that suit their particular allergies. The content of the app will be crowd sourced by fellow food allergy sufferers. A rating system will be used to indicate to the user how good the restaurant is and how well it caters to their allergies. The user will specify their allergies and the content will be tailored to their selection. The user can search for restaurants by map, to find restaurants near them, or by list to more quickly see the higher rated restaurants. The user can also review any restaurant to add to help their peers find suitable eateries. They can also add restaurants to the system through the app. All the content for the app will be stored on a web server so that any user can access it with an android device and an Internet connection.

1.3 Project Challenges

This project is an android app that incorporates many different technologies into its system. The overall challenge of the project is to integrate all the disparate elements seamlessly.

List Of Technologies Utilized

- Java
- Google Maps API
- Android GPS service
- PHP
- RESTful services
- MySQL

1.3.1 Google Maps Api

The app will utilize Google maps to indicate the location of the restaurants in the system. Map pins will be used to show locations and when the user taps pin information about the restaurant will be displayed. By using the android GPS location services the user can easily see their location on the map while searching. Also the map should zoom to the users location when first opening the map activity.

1.3.2 Web Server with MySQL

In order to have dynamic content on the app and web server with a database will need to be utilized. The data will need to be retrieved and updated using http requests. Using the principles of REST (Representational State Transfer) will help to ensure data transferred to and from the database.

2 Technologies Researched

2.1 Introduction

In this chapter I will outline the technologies I researched which were necessary to complete the app. I explored various programming languages, technologies and software to assist in the building of a RESTful android app. Some I rejected for but the following are the technologies I used to develop the project.

- Eclipse and Java
- Representational State Transfer principles
- Web Server and MySQL
- File Transfer Protocol
- TextWrangler and PHP
- Android Platform

2.2 Eclipse and Java

The Integrated Development Environment (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 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.

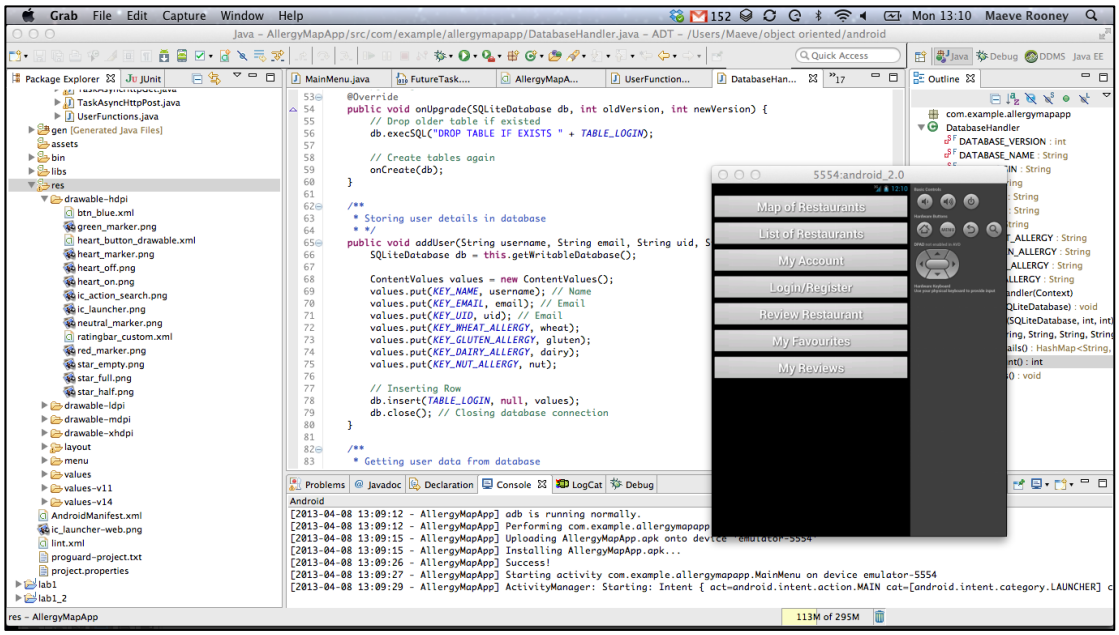


Figure 2-1: Eclipse for Mac with Android Emulator

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

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 Wi-Fi 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

2.3 RESTful Services and Android

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]. Alternatively JSON (JavaScript Object Notation) can be used which is ideal as it is easier to parse in Java. To get a JSON object in java all you have to do is pass a JSON encoded response string into the built in function "JSONObject(String)". JSON is more lightweight than XML. It uses brackets instead of tags and does not require a DTD (Document Type Definition) declaration.

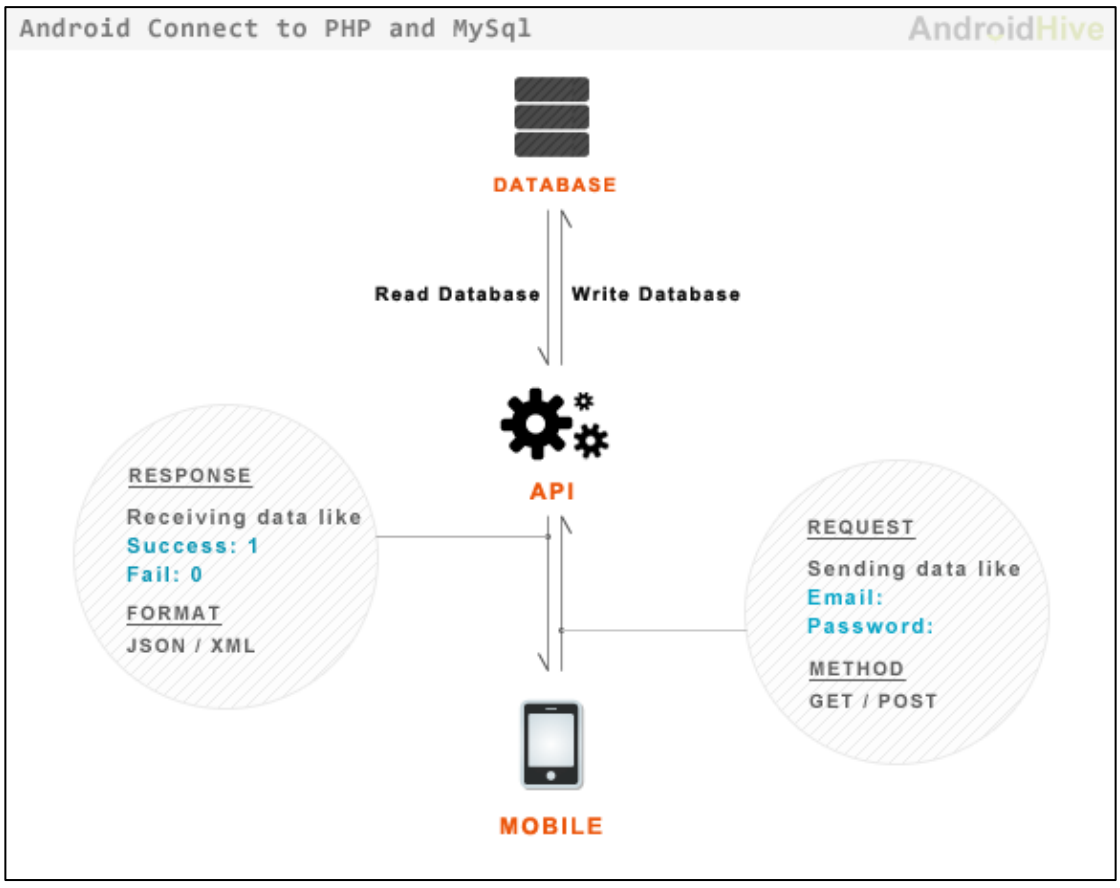


Figure 2-3: Android RESTful API

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 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.

2.4 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.

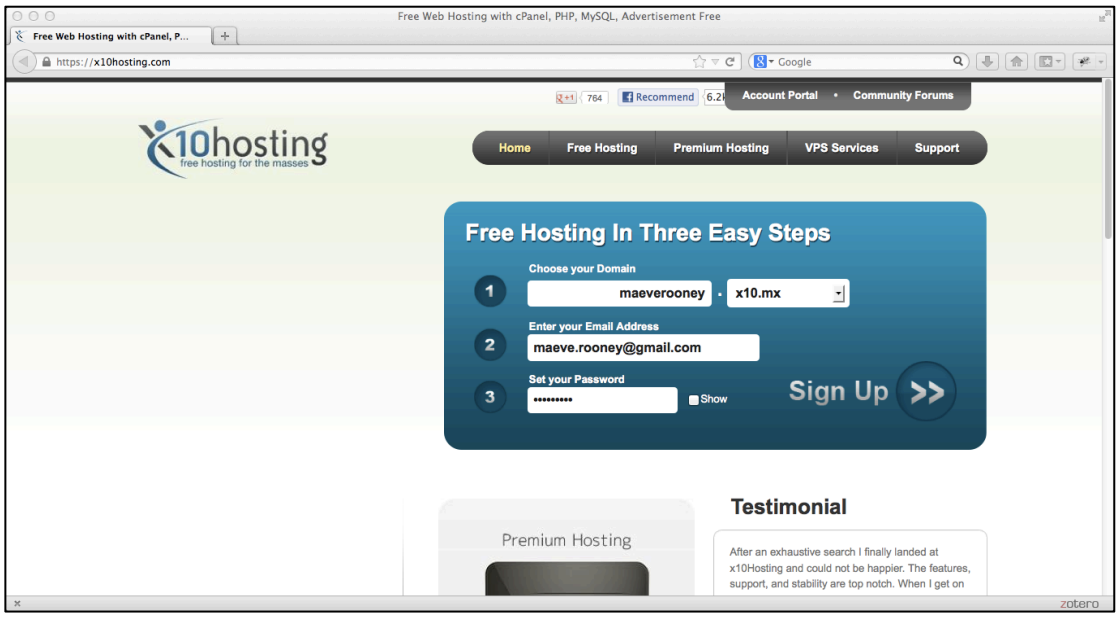


Figure 2-4: Signup page for hosting on x10hosting.com

After signing up for a free website on X10hosting.com I accessed the cPanel to create a MySQL database. In order to access this database I needed to assign a user to the database and assign permissions. I assigned one admin user with permissions to with all permissions and another general user with restricted permissions. The general user cannot create, alter or delete tables.

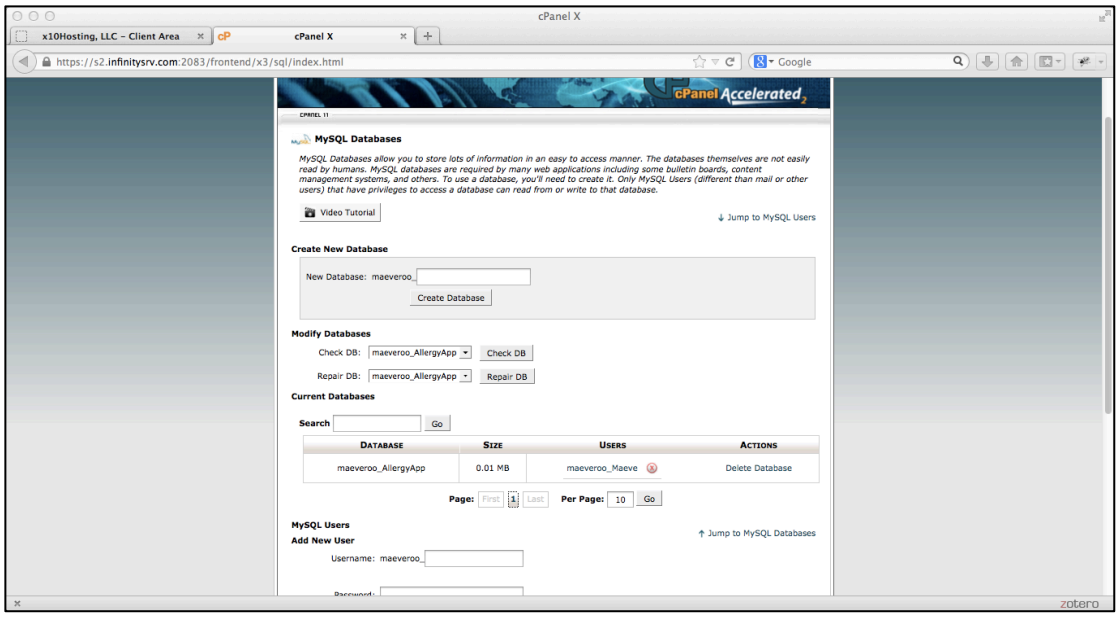


Figure 2-5 : cPanel page to manage MySQL databases

For the purposes of this project this webserver will suffice. It is free which comes with its problems such as lower reliability of service as well as timeouts during periods of high traffic to the site. This problem is not an issue while the app is not a commercial venture. However if I were to launch the app I would need a reliable service that could handle a large number of concurrent users.

2.5 TextWrangler and php, mysql

As the web server is separate from the app I wanted to develop it in a separate environment that Eclipse. All I needed was a simple text editor that understood PHP code and I found TextWrangler more than met my requirements with a simple interface and colour coding of the PHP scripts. The PHP scripts will contains all the functions to interact with the database e.g. insert and update. The will also handle the post and get requests and extract the necessary parameters.

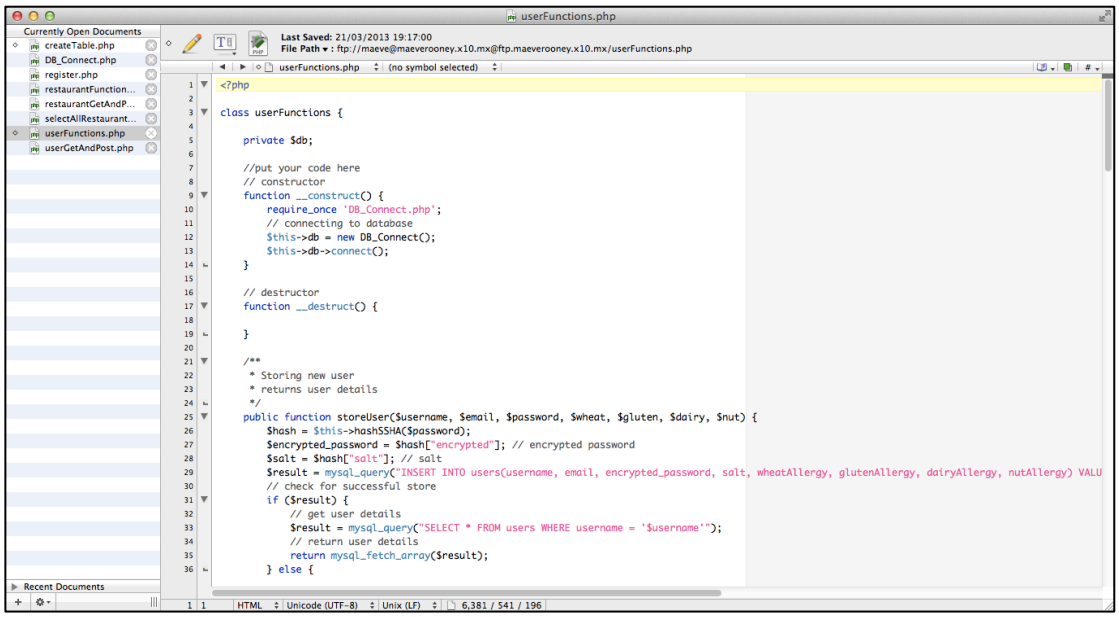


Figure 2-6: TextWrangler text editor

2.6 FTP Program – CyberDuck

To transfer the PHP scripts to the website I used a file transfer program. To make a connection over which to transfer the files the URL should start with 'ftp://' instead of 'http://'. By using a username and password to complete the connection no unauthorized users can access the file systems of the website.

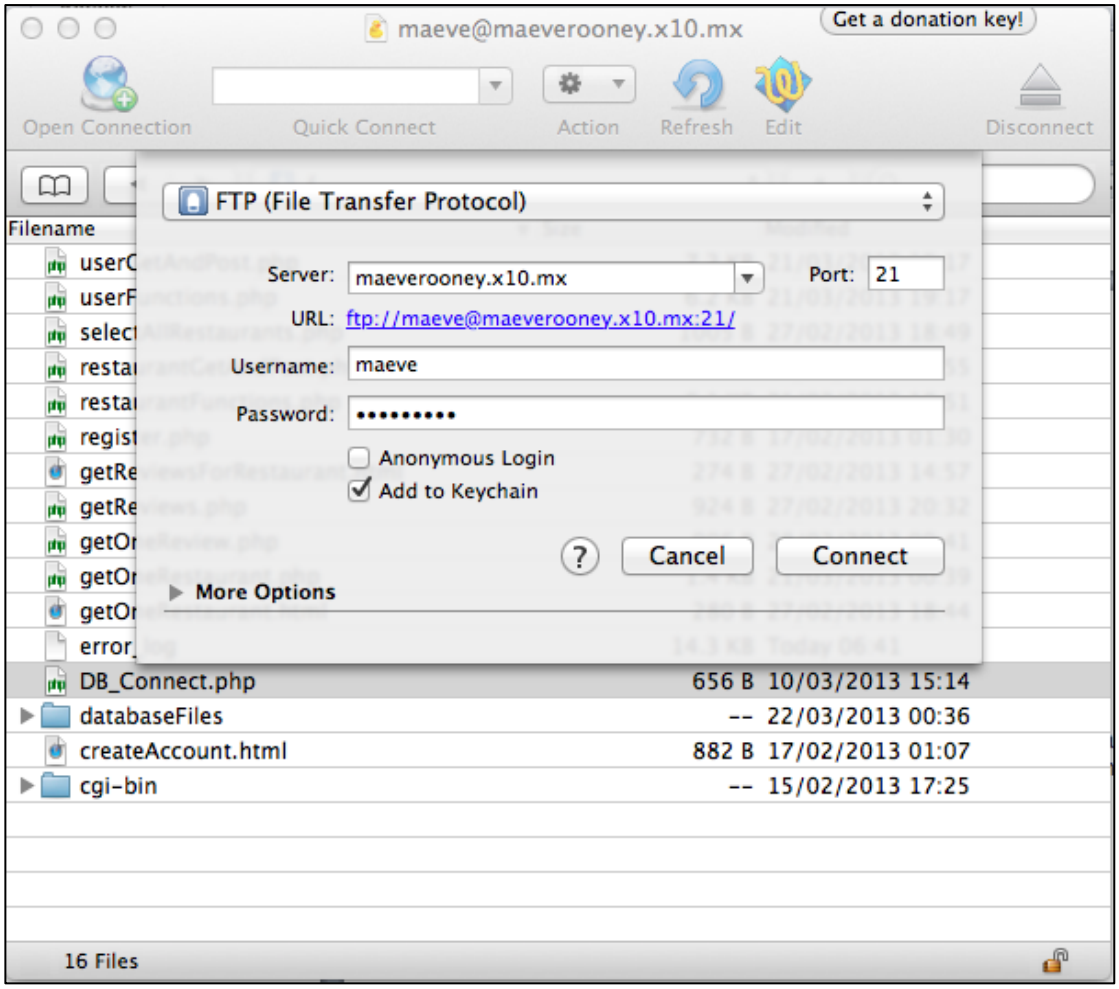


Figure 2-7: CyberDuck manages file transfer

2.7 Android platform

When it came to a choice as to which mobile platform to develop for the biggest decision factor was the programming language necessary for development. Android would require Java, iOS needs Objective-C and Windows needs Phone C#. I wanted to learn a new language to expand my knowledge so as I already knew C# that rule Windows Phone out. After a cursory look at both Java and Objective-C, I felt Java to be quicker to grasp. In addition Android has a lot of extensive development documentation online to assist developers [3].

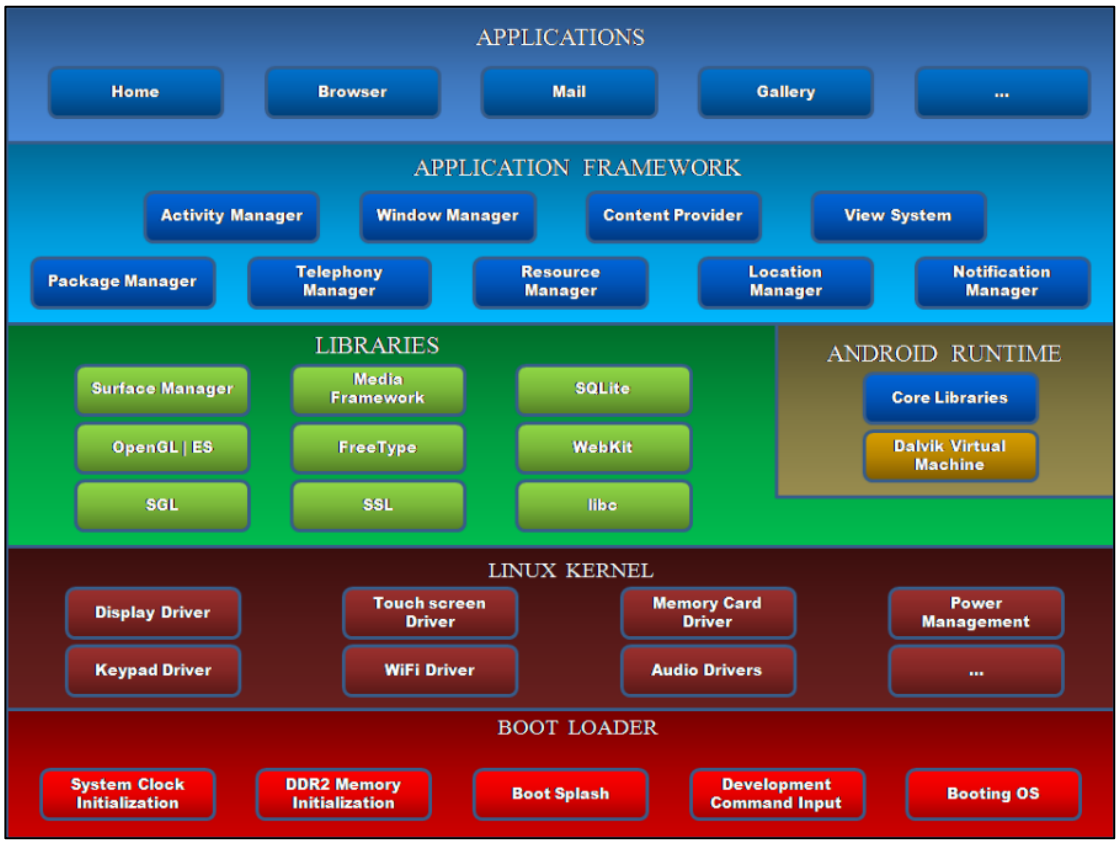


Figure 2-8: Android Architecture Diagram

I will build the app on the android platform using the application framework. Significant aspects of this framework that I will utilize are the Location Manager, Content Provider and Activity Manager. The Location Manager will give me the GPS location of the device. The Content Provider will handle access to the local SQLite database where I will store the users data on login. The Activity Manager will control the opening and closing of the various screens in the application. In other words it knows when to call onCreate() onResume() etc. in the various activities.

There are a wide variety of devices that run the android platform as well as many versions of the android platform. Developing with Eclipse and Android Development Tools forces you to develop for the most recent Android Platform. In this case the app is targeted for the version 4.1 (JellyBean API 17). It is the newest version as of January 2013 but does not have the largest market share of android devices with only 9% [4]. Version 2.3 (Gingerbread) has the largest market share with 47.4%. This is the version that my development device (Samsung Galaxy Ace) runs on. By running the app successfully on both the Samsung device and the Emulator (version 4.1) it can be hypothesized that it will run on all versions in between the two.

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.4%
2.2	Froyo	8	9.0%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	47.4%
3.1	Honeycomb	12	0.4%
3.2		13	1.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.1%
4.1	Jelly Bean	16	9.0%
4.2		17	1.2%

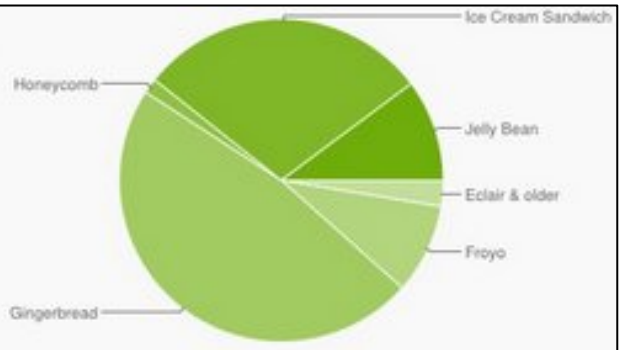
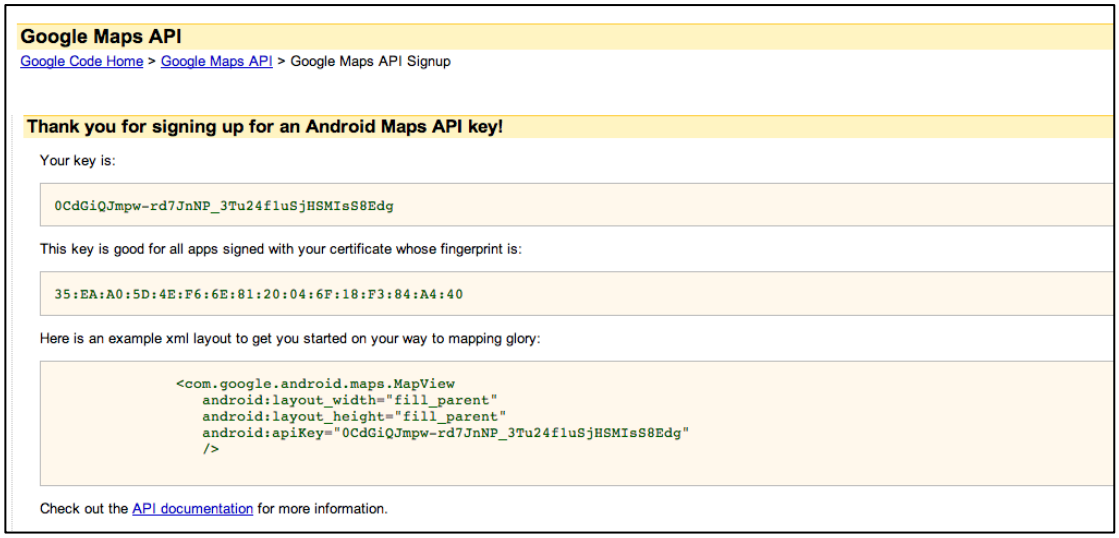


Figure 2-9: Market Share of Android Versions 2013

2.8 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.

The unique key can be retrieved from the Google Developer Website [5].



Google Maps API

Google Code Home > Google Maps API > Google Maps API Signup

Thank you for signing up for an Android Maps API key!

Your key is:

```
0CdGiQJmpw-rd7JnNP_3Tu24f1uSjHSMIsS8Edg
```

This key is good for all apps signed with your certificate whose fingerprint is:

```
35:EA:A0:5D:4E:F6:6E:81:20:04:6F:18:F3:84:A4:40
```

Here is an example xml layout to get you started on your way to mapping glory:

```
<com.google.android.maps.MapView
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:apiKey="0CdGiQJmpw-rd7JnNP_3Tu24f1uSjHSMIsS8Edg"
   />
```

Check out the API documentation for more information.

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

And then this key should be inserted into the xml file that controls the layout for the map activity screen.

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/maptablayout"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <com.google.android.maps.MapView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_above="@+id/LinearLayout02"
        android:apiKey="0CdGiQJmpw-rd7JnNP_3Tu24f1uSjHSMIsS8Edg"
        android:clickable="true" />

    <LinearLayout
        android:id="@+id/LinearLayout02"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_alignParentBottom="true">
            <Button
```

Figure 2-11: Inserting Google API key into map layout xml

3 Design & Methodology

3.1 Introduction

This chapter outline the design of the app. It explains which design methodology was chosen and why. The various use cases showing the interactions between the user and the app are illustrated. I will outline the database design and the overall system architecture.

3.2 Use Case Design

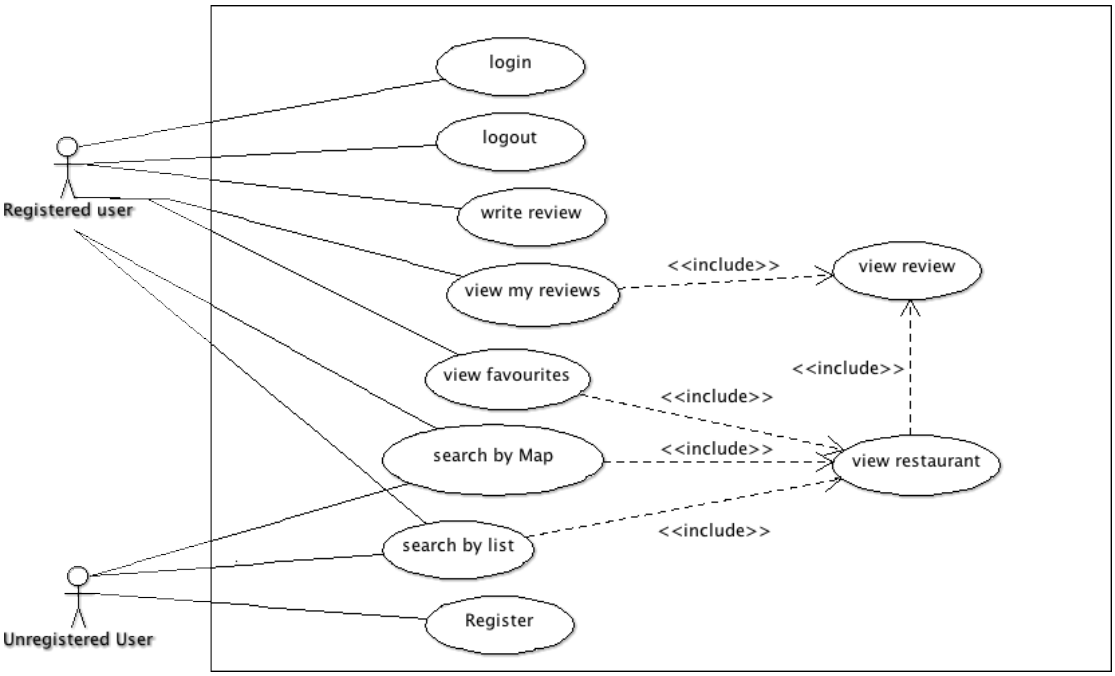


Figure 3-1: Use Case For User

This use cases focus on how the user interacts with the app. There are two kinds of user. An unregistered user can register but cannot log in, log out or write a review. All users can search for restaurants by map or by list. This is a high level view of the system looking at all the possible activities the user can do.

3.3 Class Diagram

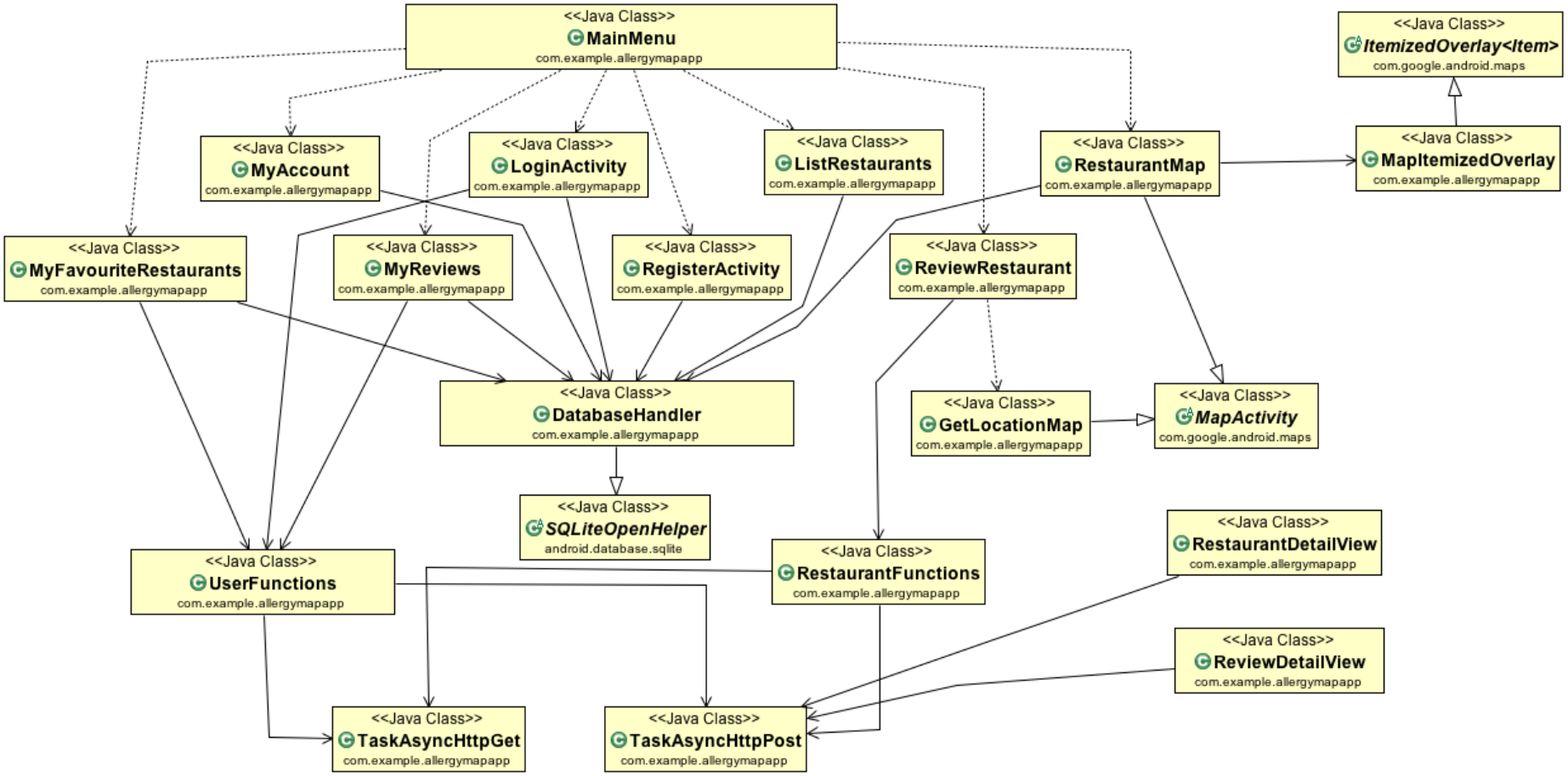


Figure 3-2: Class Diagram

Most of the classes in the system are activities that control the views. Other class's handle retrieving data from and pushing data to the server, interfacing with map overlay objects and handling the local database.

Activity Classes

Name	Function
MainMenu	List of buttons to navigate to the other activities
MyAccount	Allows user to see their current details and change them. For example change email address
LoginActivity	If user is logged in they are presented with a dialog to logout. Otherwise they have to enter their username and password to login
RegisterActivity	Asks the users to enter details to register account. Checks availability of unique username and password.
ListRestaurants	Displays all the restaurants in the system in a list view
RestaurantMap	Displays all the restaurants in the system on a Google Map
ReviewRestaurant	View to both add and review a restaurant. Checks if restaurant exists already and modifies view to reflect result.
MyFavouriteRestaurants	A list view of all the restaurants the user has favourited
MyReviews	A list view of all the users reviews
RestaurantDetailView	Display the details of a chosen restaurant such as phone number and address as well as a list of the reviews of that restaurant
ReviewDetailView	Displays an individual review including the authors username and the various ratings they gave

Table 1: Activity Classes

Other Classes

Name	Function
MapItemizedOverlay	Handles the map pins that are map overlay objects. For example when a pin is tapped an alert box is displayed
DatabaseHandler	Controls the interaction with the local database where the current users details are stored. When user logs in a row is inserted to the user table. When they log out this row is deleted.
UserFunctions	Handles the functions to post to the user table on the web server database. For example loginUser() and changeUserDetails().
RestaurantFunctions	Handles the functions to post to the restaurant and review table on the web server database. For example addRestaurantToDB() and addReviewToDB()
TaskAsyncHttpGet	Makes http get requests asynchronously from the main user interface thread.
TaskAsyncHttpPost	Makes http post requests asynchronously from the main user interface thread.

Table 2: Other Classes

3.4 Design Methodology

3.4.1 Sprial Methodology

The Spiral methodology combines the features of the Waterfall and Prototyping Incremental models [6]. It is perfect for a beginner like me because I can start with a small understanding of the scope of the project and expand this understanding and scope as my abilities and knowledge increase. Each iteration of the spiral methodology involves producing complete prototypes of the project until the final prototype is developed.

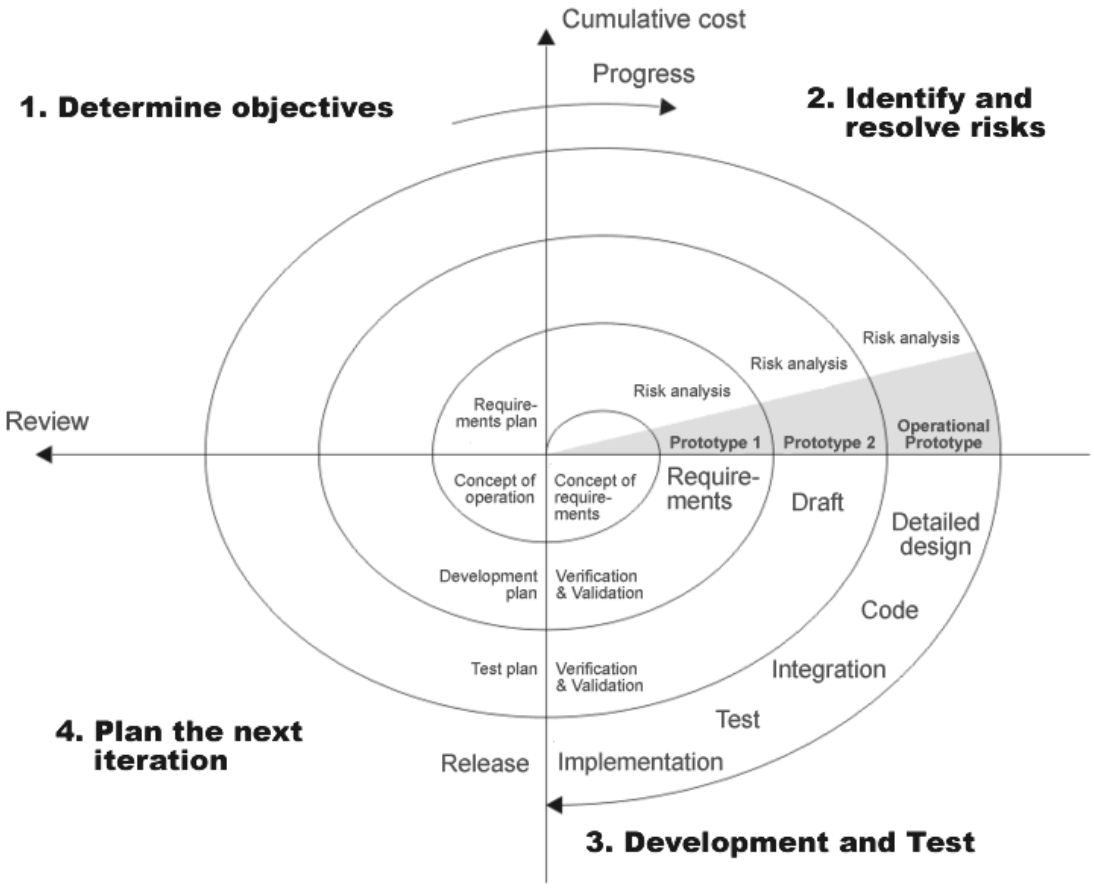


Figure 3-3: Spiral Methodology

3.4.2 Developing With Spiral Methodology

The four main stages of a cycle of the spiral methodology [7]:

1. Determine objectives

   Outline what is necessary to complete this iterative of the spiral including tools and skills needed.

2. Identify and resolve risks

   Evaluate alternatives and identify and resolve risk issues.

3. Development and test

   Develop and verify the project for this iteration. This is where all the coding, compiling, running and testing happen.

4. Plan the next iteration

   Review the current iteration and plan what further improvements and features to add in the next iteration

3.4.3 Increments

1. Coming from a complete beginners perspective when it come to mobile app development the first increment was simply to ask the user to input a word and then display that word on the tap of a button.
2. Get a Google Map running on the device with a map pin
3. Set up web database with restaurant information including GPS location. Then call this information from the app and display restaurant locations on map.
4. Set up Register and Login system including user table in web database. Password encrypted when registered and verified on login. Use AJAX principles to verify uniqueness of email address and username when registering account.
5. Set up Main menu to direct user to various activities. Place tab buttons at the bottom of all screens to navigate user to main pages.
6. Set up list activity to display restaurants in system as a list.

7. Tapping map pin and list item bring user to new activity to display restaurant information including list of reviews. Tap on review to bring user to view of review information.
8. Set up review activity including rating bar with stars. Password required to post review to database.
9. Set up map view to place pin where new restaurant is located. Retrieve the location data from this pin placement and add to restaurant details in database.
10. Check that user is logged in in all activities to tailor display results to their settings. Only display results for allergies that affect them.
11. In Map view get ratings for restaurants. If ratings are good for the users allergies display with a green map pin. If they are negative display with a red map pin. If the user is not logged display all restaurants with a blue map pin.
12. Allow user to favourite/un-favourite a restaurant when user taps a map pin or list item. Change the map pin of a favourite restaurant to a pink pin with a heart.
13. Create activities to list users favourite restaurants and reviews.

3.5 Web Server and MySQL Design

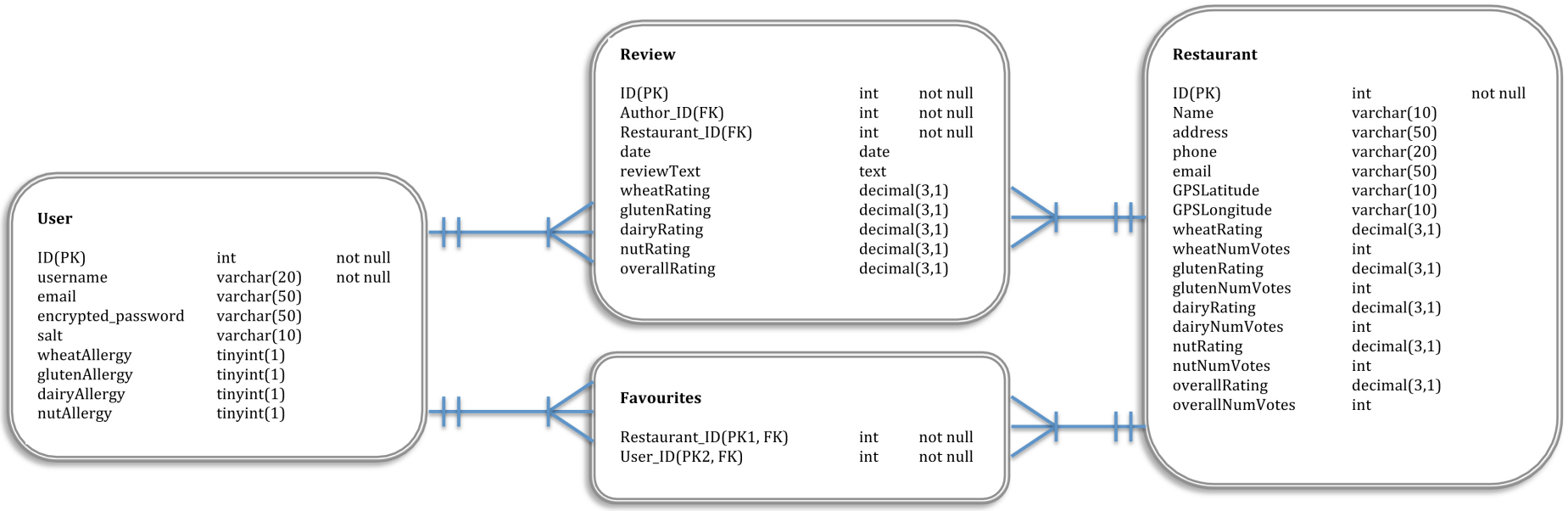


Figure 3-4: Entity Relation Diagram For Web Database

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	

Table 3: Web Server User Table

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	

Table 4: Web Server Restaurant Table

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	

Table 5: Web Server Review Table

Favourites Table

FIELD	TYPE	SIZE(byte)	NULL	DEFAULT
User_ID (PK1, FK)	int	4	no	
Restaurant_ID (PK2, FK)	int	4	no	

Table 6: Web Server Favourites Table

3.6 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	

Table 7: Local User Table

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 (e.g. 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.

3.7 User Interface Design

In the early stages of the project I mocked-up some screens for the app to wrap my head around the various activities needed to build the app. This step was made easy using the free edition of 'Prototyper' [8]. It comes with widgets of buttons and sliders etc. to easily build prototypes. The screens I developed using this tool are shown in the Appendix section 9.2. The first implementation of these screens, on an android device are shown in Appendix section 9.3.

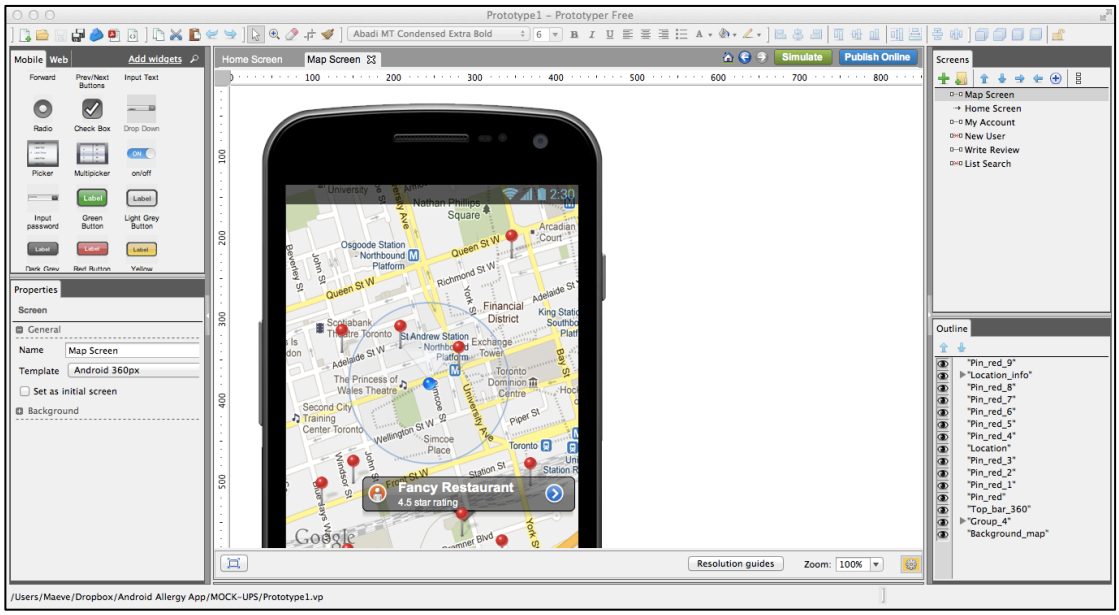


Figure 3-5: Prototyper Programme to mock up screens

The app should be very easy for the user to navigate through and the location of the various activities should be intuitive and follow a natural work-flow. All the activities feature a navigation menu at the bottom that features four buttons. A “Back” button to return to the previous activity, a “Home” button to go to the main menu, a “Map” button to go to the Restaurant Map Activity and a “List” Button to go to the Restaurant List Activity. These buttons make it easy for the user to quickly get to the activity they need.

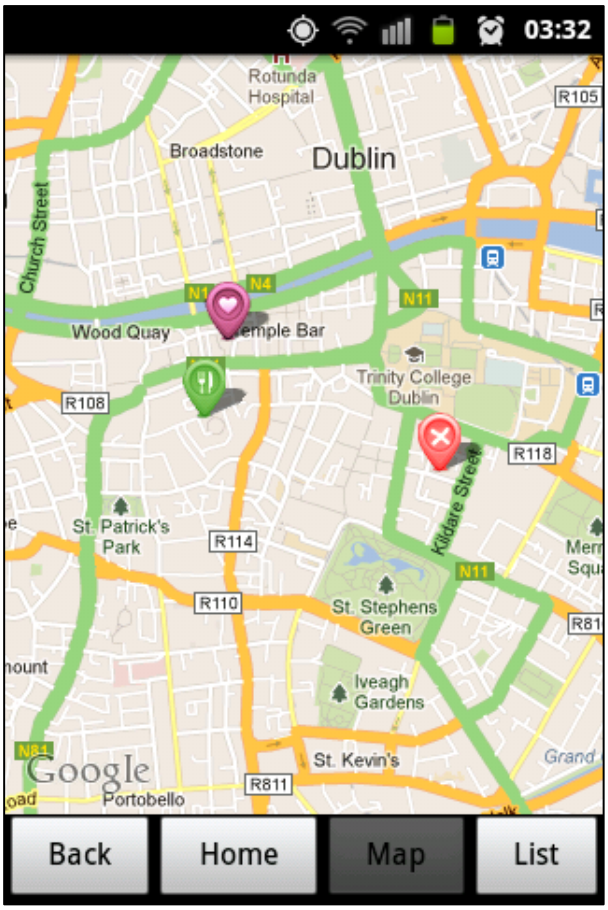


Figure 3-6: Navigation menu at the bottom of all screens

3.7.1 Navigation Diagram

Each rectangle symbolizes and activity in the app. The central eight activities can be navigated to via buttons on the home screen. The maximum depth of activities from the main screen is two. This means that a user can get to any screen with a maximum of three clicks. For example to get to the “User Review” screen a user can click on the “My Favourites” Button and then a click on one of the restaurants listed to go the “Restaurant Info” screen. Here a final click on one of the listed reviews takes the user to the “User Review” screen.

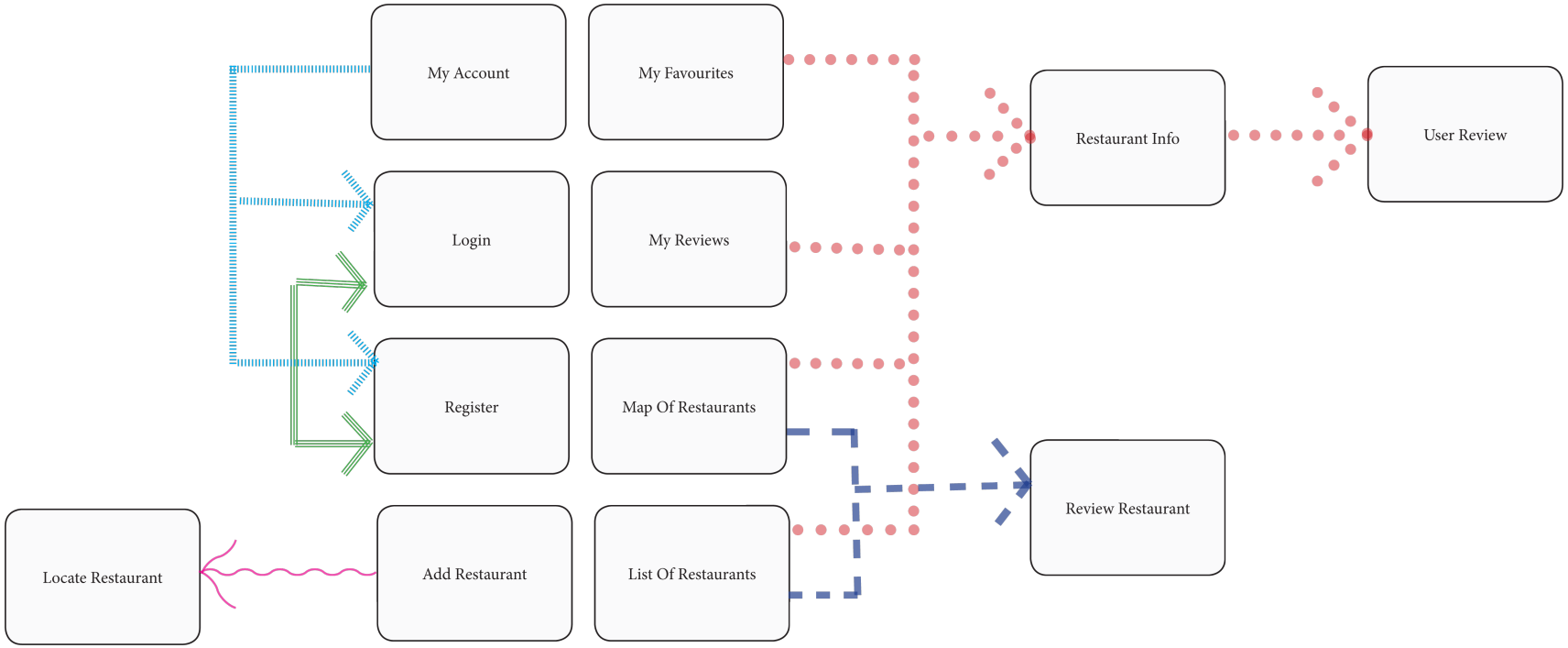


Figure 3-7: Navigation Diagram

3.8 List of Features

1. Register an Account
2. Login
3. Logout
4. Modify Account
    a. Change username
    b. Change email address
    c. Change password
    d. Change allergies
5. Search for restaurants by map
6. Search for restaurants by list
7. Add a restaurant
    a. Locate restaurant on map
    b. Rate restaurant for each of my allergies
    c. Give restaurant overall rating
    d. Write your opinion on experience in restaurant
8. Review a restaurant
9. View Details of restaurant including reviews for that restaurant
10. View Individual Review
11. Save a restaurant to ‘My Favourite Restaurants’
12. Remove a restaurant from ‘My Favourite Restaurants’
13. View ‘My Favourite Restaurants’
14. View ‘My Reviews’

4 Architecture & Implementation

4.1 System Architecture

The Food Allergy Map system architecture is a five-tier system. The tiers comprise of the normal three tiers of an application – the presentation tier, the application tier and the data tier. The fourth tier utilizes the android platform location manager to create the Android GPS Tier. The fifth tier incorporates the Google Maps system to create the Google Maps Tier.

4.1.1 Presentation Tier.

This tier comprises all the screens of the app. These screens handle out the user interacts with the system. This tier is implemented with activity classes in Java.

4.1.2 Application Tier.

The application tier is where all the background computation occurs. It is also the layer that controls the transfer of data from the data tier to the presentation tier. This included http request to the web database and queries to the local SQLite database.

4.1.3 Data Tier.

The data tier is where all the content for the app is stored. The content is dynamic so the data must be retrieved regularly to reflect the current content. This app has both a local database and a web database.

4.1.4 Android GPS Tier.

This tier uses the Android platform application framework 'Location Manager' and the GPS sensors of the device to get the users location. Android uses satellites and triangulation to get location information of the device.

4.1.5 Google Maps Tier.

The final tier is built using Google Maps API. This is a ready-made tool to zoom around and interact with objects placed on an accurate street map.

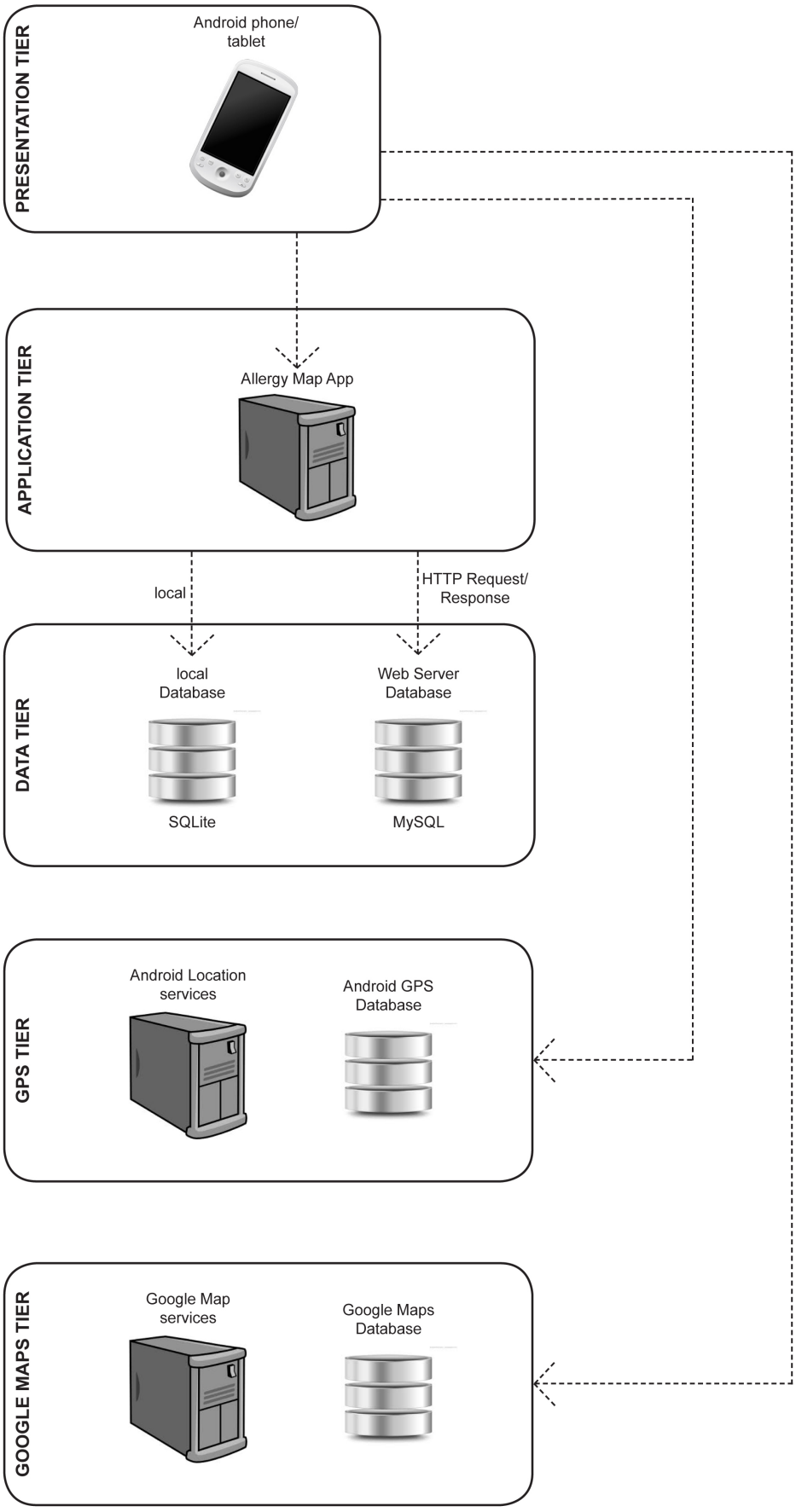


Figure 4-1: Final System Architecture

4.2 Key Developments Components

This project integrates several components to run effectively. The following are the key developments components:

1. Setup of the Activities
2. Setup of Web Server with MySQL Database
3. Making Http Requests from Device to Web Server
4. Validation of Login and Registration
5. Running Google Maps API in app
6. Using Rating Bars to Review Restaurant

These areas are further explained in the following sections.

4.3 Setup of The Activities

In order for a Java class to act as a presentation screen for an android device it must extend and Android activity.

```
import android.app.Activity;
public class MyAccount extends Activity{ }
```

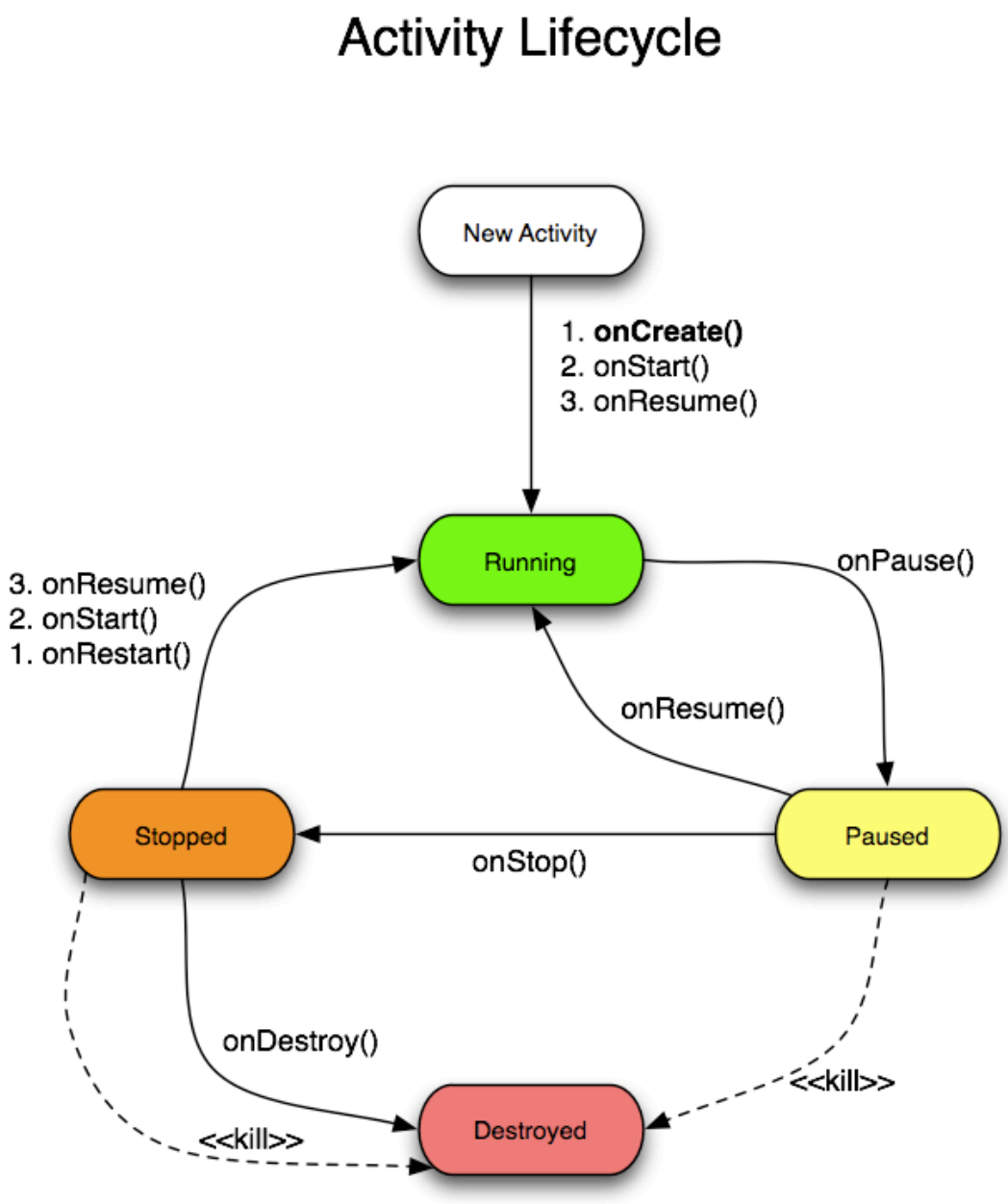


Figure 4-2: Android Activity Lifecycle

The Android Activity is not controlled by the application but by the Android runtime environment [8]. When a user starts an activity the android lifecycle calls a set of methods to change the state of the activity to active and visible. Then when the user starts a new activity the state is changed to invisible but the instance of the activity remains intact so that the user can navigate back to it as they left it. The Android runtime environment control which method in the activity to call and when.

4.3.1 Activity Lifecycle Methods Explained [9]:

1. onCreate() : When an Activity is first launched the onCreate() method is called. This is where the User Interface is created and variables are initiated.
2. onStart(): This method is called before the Activity is made visible to the user.
3. onResume(): This is when the Activity become visible and Active for the user to interact with.
4. onPause(): This method will be called when the system is about to resume another Activity on top of this one or when the user is about to navigate to some other parts of the system. Here is where you should save data which is critical to the system before the activity closes. It is called for example when the device screen turns off. To reopen the activity onResume() is called.
5. onStop() This is called the user has presses the back button. To reopen this to this activity onCreate(), onStart()and onResume() will need to be called.
6. onDestroy(): When this method is called the Activity is destroyed. This is the final method we can call before the Activity is destroyed. This occurs either because the Activity is finishing the operation or the system is temporarily destroying it to save space.

To create an Activity you must override these methods as necessary. Usually it is only necessary to override the onCreate() method. All other methods will inherit from the parent class.

4.4 Setup of Web Server with MySQL Database

As explained in section 2.3, I signed up for web hosting with X10hosting.com and set up a MySQL database on this site called ‘maeveroo_AllergyApp’. I also set up a user called ‘maeveroo_Maeve’ with a password that I use to connect to this database. I use PHP scripts to connect to the database that are stored on the same server as the database. Therefore I can use ‘localhost’ to create a connection to MySQL before selecting my database to query.

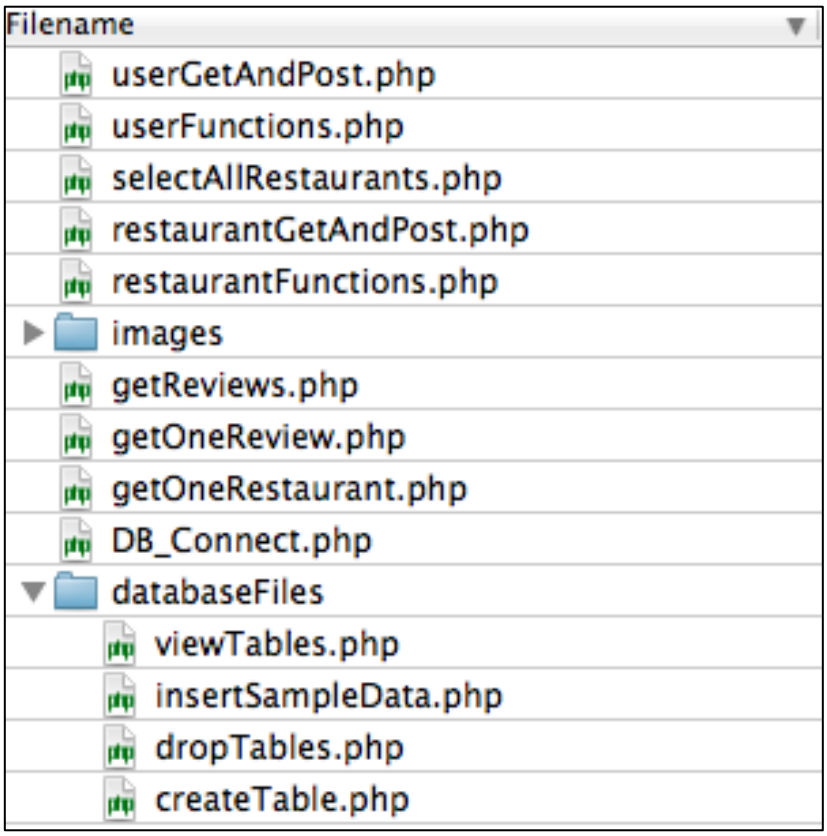


Figure 4-3: File Structure of Website

```php
<?php

class DB_Connect {

    // constructor
    function __construct() {
    }

    // destructor
    function __destruct() {
        // $this->close();
    }

    // Connecting to database
    public function connect() {
        // connecting to mysql
        $user="maeveroo_Maeve";
        $password="password";
        $database="maeveroo_AllergyApp";
        $con = mysql_connect(localhost,$user,$password);
        // selecting database
        @mysql_select_db($database) or die( "Unable to select database");

        // return database handler
        return $con;
    }

    // Closing database connection
    public function close() {
        mysql_close();
    }
}

?>
```

Figure 4-4: PHP script to Connect to Database

After connecting to the database I can run queries such as create table, alter table and insert into table queries. To run the queries that the user would make to the database I created a script containing functions. This script contains functions such as StoreUser() and GetReviewForRestaurant(). I then created a script to handle http get and post requests to the website. This script checks the parameters in the request and determines which function to call from the function script. It then returns the result of the query in an array as a JSON encode response string.

In order to run the app for testing purposes I inserted sample data to the database. This data includes users with various allergies, restaurants, reviews and restaurants favourite by the mock users.

4.5 Making Http Requests from Device to Web Server

The Web Site is setup to respond to requests with JSON encoded strings. Now the app needs to make request to get this data and know how to handle the response.

To add parameters to a URL in java I used a list of basic name pair values. In order to indicate to the server which function to perform a parameter named ‘tag’ is included. This ‘tag’ value varies for each function. For Example when logging a user in the tag value is ‘login’, when registering a user the value is ’register’ and to review a restaurant the value is ‘review’.

```
List<NameValuePair> params = new ArrayList<NameValuePair>();
 params.add(new BasicNameValuePair("tag", login_tag));
 params.add(new BasicNameValuePair("username", username));
params.add(new BasicNameValuePair("password", password));
```

These parameters are then passed to the class that handles Http Requests Asynchronously where they are added to the http request.

```
HttpPost httppost = new HttpPost(urlString);
httppost.setEntity(new UrlEncodedFormEntity(params));
```

Then the response string is retrieved when the request is executed.

```
TaskAsyncHttpPost httpRequest = new TaskAsyncHttpPost(params, mContext);
String response = httpRequest.execute(loginURL).get();
```

This code is surrounded by a try/catch block as the request might fail if the server is down which will cause the app to crash. The final step in making a request to the server is to turn the response string into a JSON object so that it can be parsed for presentation to the user.

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

4.6 Validation of Login and Registration

When a user registers the must enter a unique email and username. They must also enter a password twice to confirm that they match. In order to assure that the user does not enter a email or username that is already in the database I use the principles of AJAX to check their input before the submit the registration form.

```
UserFunctions userFunction = new UserFunctions(RegisterActivity.this);
JSONObject json =
        userFunction.checkUsername(inputUsername.getText().toString());
        if (json.getString(KEY_ERROR) != null) {
                usernameErrorMsg.setText("Username unavailable");
        }
```

If the entered username or email already exists in the database an error message is displayed and the form will not submit. When they re-enter a valid input the error message is removed and the user is able to submit the form.

As well as seeing it the email is unique the system also checks to see if it is a valid email address. This is done by comparing the entered email address to a regular expression for email addresses.

```
if (!inputEmail.getText().toString()
        .matches("[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+") && s.length() > 0){
             emailErrorMsg.setText("invalid email");
}
```

To see if the passwords match the two entered passwords are compared. If they do not match an error message is displayed and the form will not submit.

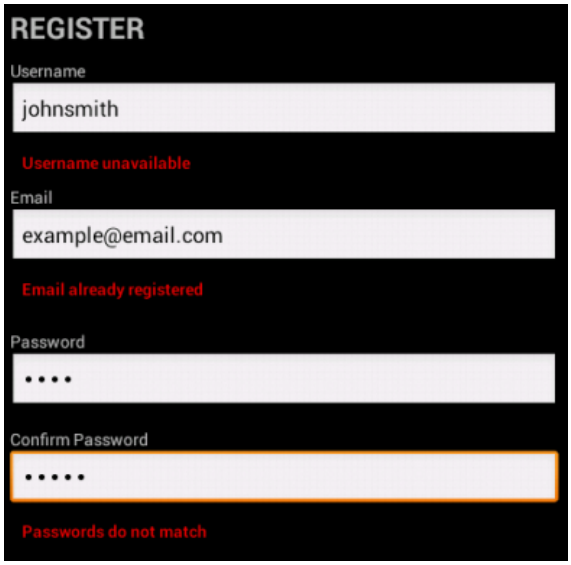


Figure 4-5: Screen Shot of Invalid Registration Input

The same checks happen in 'My Account' activity when the user wants to change their username, email or password. Again, they cannot submit invalid options to the database.

When a user logs in with their username and password the web server validates them. The password is encrypted using the 'salt' and compared to the encrypted password in the database where the username matches the one entered. If the comparison is correct then a successful response is returned and the login is successful. Otherwise if the comparison fails or if the username is not found then an error response is returned and the login fails.

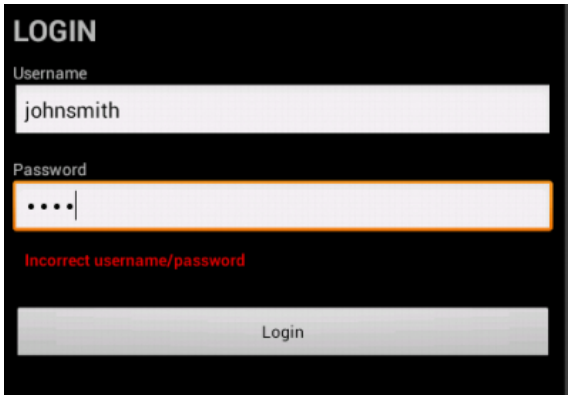


Figure 4-6: Screen Shot of Login Fail

4.7 Running Google Maps API in app

I obtained a Google Maps API key and inserted it into the map view layout xml file as described in section 2.8. By doing this I can create an Activity which shows a Google map [10].

```
import com.google.android.maps.MapActivity;

public class RestaurantMap extends MapActivity {
       //Set the view to the map_view xml with the api key
      setContentView(R.layout.map_view);
      MapView mapView = (MapView) findViewById(R.id.mapview);
      //define a map pin image
      Drawable map pin =
              this.getResources().getDrawable(R.drawable.map_marker);
    //Set this image as a map overlay item
       MapItemizedOverlay itemizedOverlay =
              new MapItemizedOverlay(drawable, this);
```

MapItemizedOverlay is a class I created to handle the map pins. When each pin for a restaurant is created it is added to this class. When a pin is tap this class identifies the pin and opens up a dialog box unique to each pin with information about the restaurant and buttons to navigate to other activities.

After creating map and the MapItemizedOverlay instance I make a call to the web database to get all the restaurants. I then loop through these items and create an overlayItem for each restaurant and add this overlayItem to the instance of the MapItemizedOverlay class. An OverlayItem takes three parameters, a Geopoint with latitude and longitude coordinates, a title string and a snippet string. The Geopoint is used to locate the pin on the map. When the OverlayItem is tapped a dialog box pops up, showing the title and the snippet.

```
//create overlay item
GeoPoint point = new GeoPoint(Latitude, Longitude);
OverlayItem overlayitem = new OverlayItem(point, “title”, ”snippet”);
//add the overlay item to the MapItemizedOverlay instance
ItemizedOverlay.addOverlay(overlayitem);
```

To implement the MapItemizedOverlay class it inherits from the Google Maps class 'ItemizedOverlay'. Its main property is an ArrayList of OverlayItems. It has a function onTap() to interact with the map pins (OverlayItems).

```
import com.google.android.maps.ItemizedOverlay;
public class MapItemizedOverlay
      extends ItemizedOverlay<OverlayItem> {

            private ArrayList<OverlayItem> mOverlays =
                  new ArrayList<OverlayItem>();

            @Override
            protected boolean onTap(int index) {
                  OverlayItem thisItem = mOverlays.get(index);
            }
```

In the onTap() function I create an alert dialog box which displays the name of the restaurant, the rating of the restaurant using stars and buttons to review the restaurant and see review for the restaurant. If the user is logged in the dialog box will also allow them to add or remove the restaurant from their favourites.

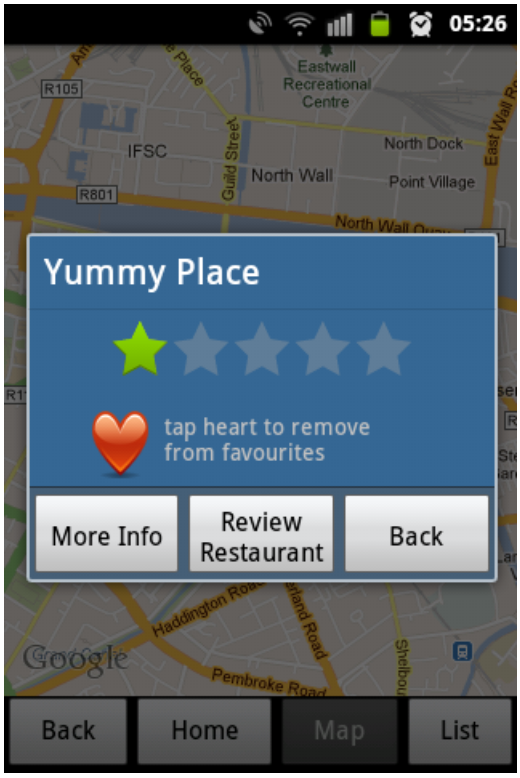


Figure 4-7: Dialog Box When User Taps Map Pin

4.8 Using Rating Bars to Review Restaurant

I needed simple user interface for the user to review restaurants. Each user review consists of the users written opinion of their experience, their overall rating of the restaurant as well a s rating for how the restaurant catered for each of their particular allergies. The text element of the review can be easily implemented with a simple text field. A text field could be used to accept a numeric text value from the user for the various ratings but this would not be as user friendly as possible. Users may try to enter invalid values.

A better solution would be to use the built in android widget of rating bars to implement the user rating system. This way it is clear to the user how to rate the restaurant and they cannot enter incorrect values.

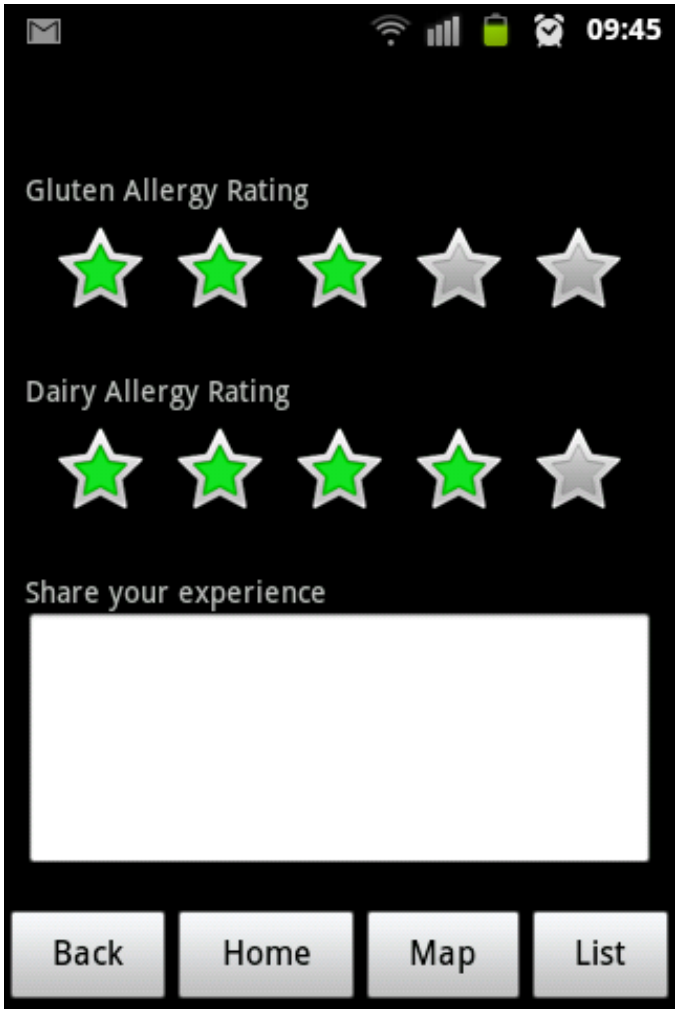


Figure 4-8: Review Screen with rating bars

To implement the rating bars they must first be initiated in the xml layout file for the review activity. Here you can specify the number of stars in the rating bar and the step size. I chose a step size of “0.5” which means that half star ratings are allowed such a 3.5 out of 5.

```
<RatingBar
        android:id="@+id/ratingbar_wheat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:numStars="5"
        android:stepSize="0.5"/>
```

Then this rating bar is retrieved in the activity class and an event listener is assigned to it.

```
float wheatRating = 0;
final RatingBar wheatRatingBar =
        (RatingBar)findViewById(R.id.ratingbar_wheat);
wheatRatingBar.setOnRatingBarChangeListener(
        new OnRatingBarChangeListener() {
        public void onRatingChanged(
                RatingBar ratingBar, float rating, boolean fromUser
                        wheatRating = rating;
                }
        });
```

When the user interacts with the rating bar the rating float is changed accordingly. Then when the “Save Review” button is pressed the rating for each attribute is posted to the web database along with the review text.

A rating bar used as an indicator is used to display ratings to the user in the other activities. The user can not interact with these rating bars as they are set up in the xml layout file as indicators and are not selectable.

```
<RatingBar
         android:id="@+id/ratingbar_Indicator"
         style="?android:attr/ratingBarStyleIndicator"
         android:isIndicator="true"
         android:numStars="5"
         android:stepSize="0.5"
         android:textIsSelectable="false" />
```

5 System Validation

5.1 Testing

In order to prove that the app behaves as expected it is important to test it. My system for this app consists of an android app and a website with database. All aspects of the system should be tested. I used the use case to devise possible scenarios to test. The system should react to user interaction as expected. This applies to both correct and incorrect input. While an installed app can only have one user at any given time many users can make calls to the central website at the same time. Therefore the website should be tested for how it handle many concurrent users.

5.2 Testing Methodologies

5.2.1 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. Test cases should be written to test invalid inputs from the user to see how the system handles incorrect use.

5.3 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
MyAccount gets user details from SQLite	MyAccount, DatabaseHandler

Table 8: Scenarios Tested

5.4 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].

5.4.1 Robotium

Robotium is a third party library that extends the android test framework [3]. 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 Activities 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.

5.5 Performance and Stress Testing of Web Server

5.5.1 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]. Test cases can be executed from the shell or by using a GUI. The application is python based and the test cases are written using XML.

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

```
<testcases>
      <case>
        <url>http://maeverooney.x10.mx/selectAllRestaurants.php</url>
        <method>GET</method>
      </case>
</testcases>
```

Figure 5-1: - Test Case of GET method

```
<testcases>
      <case>
          <url>http://maeverooney.x10.mx/userGetAndPost.php</url>
          <method>POST</method>
          <body><![CDATA[tag=login&username=johnsmith&password=secret1]]></bod
          <add_header>Content-type: application/x-www-form-urlencoded</add_hea
      </case>
</testcases>
```

Figure 5-2: Test Case of POST method

When running the test cases 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)

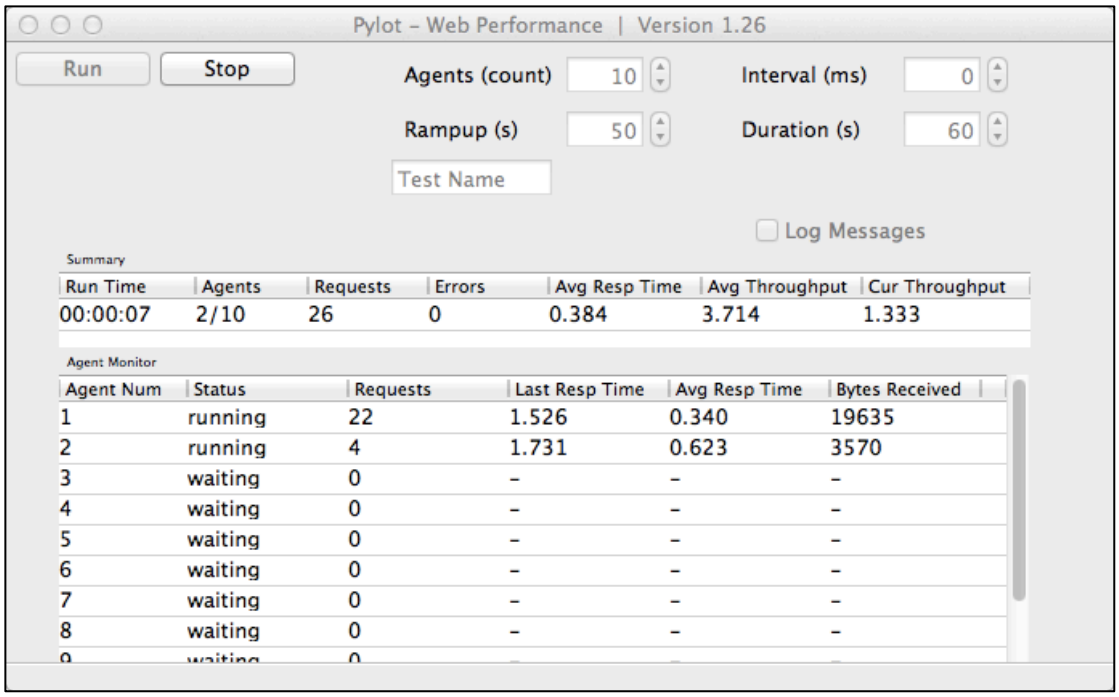


Figure 5-3: Pylot GUI running test

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.

Number 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 9: 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.

5.5.2 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 seconds

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

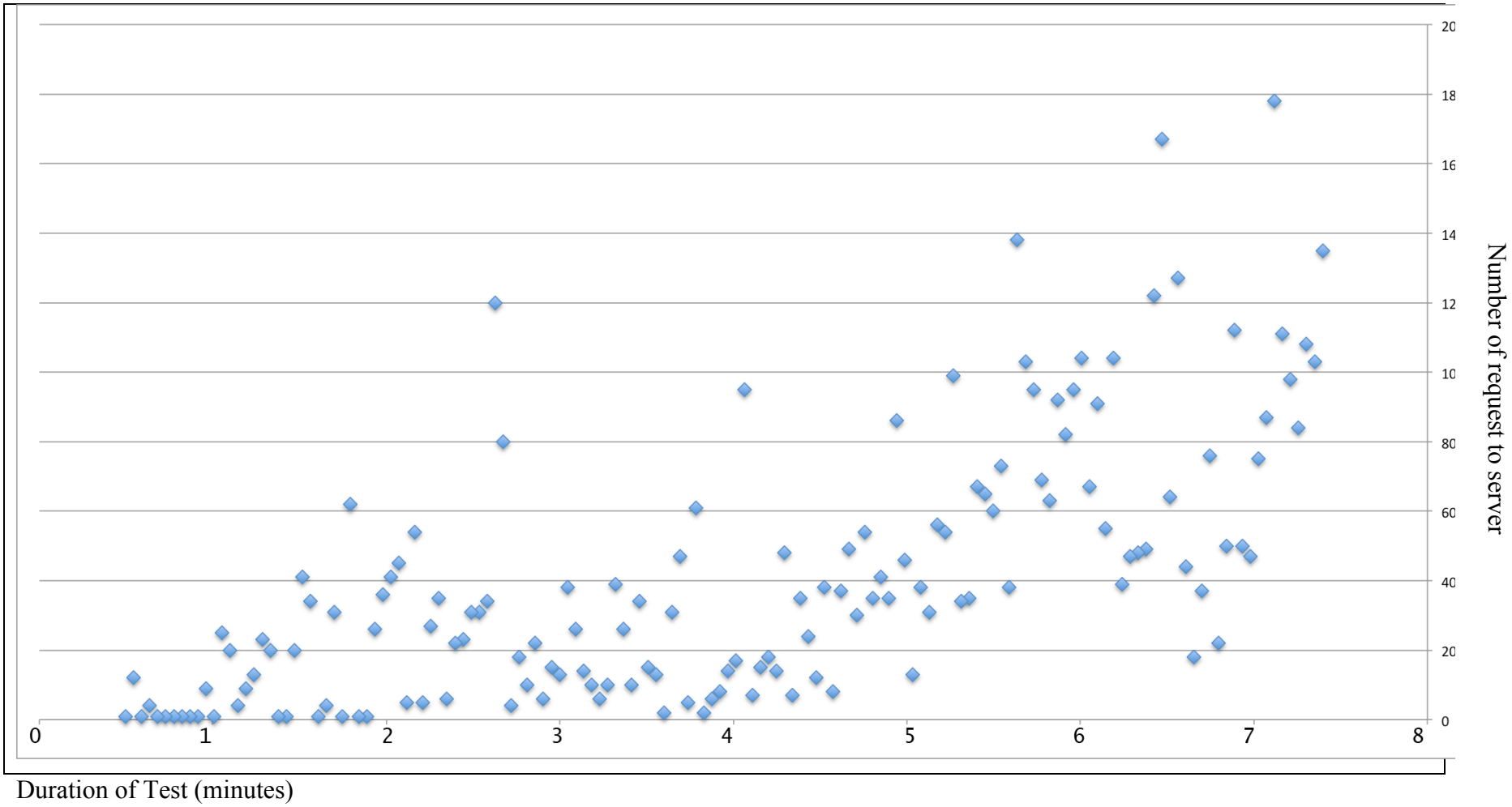


Figure 5-4: Number of requests per minute

5.6 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.

6 Project Review

6.1 Original Plan

As this was the first software project I developed I kept my original plan for the app relatively small. By using the Spiral Methodology I could build upon this plan as the project commenced and my understanding developed. The initial plan was to implement a few key features as follows:

1. Map of restaurants in database
2. List of restaurants in database
3. Activity to review restaurant
4. Activity to view a review
5. Activity to add a restaurant to the database
6. User account with registration/login

6.2 New Features

As the project evolved through the various iterations of the spiral methodology I added further features to improve the usability of the app for the user.

1. User account allows user to select their allergy/allergies from a set list
2. User can ‘favourite’ a restaurant
3. Ratings only relevant to users allergies show when user is logged in
4. Different coloured icons on map used to show restaurants with good and bad reviews as well as users favourites
5. Activity to list users favourites
6. Activity to list users reviews

6.2.1 Improvements To Map Interface with Icons

Figure 6-1: Mock-Up with Simple Map Pins

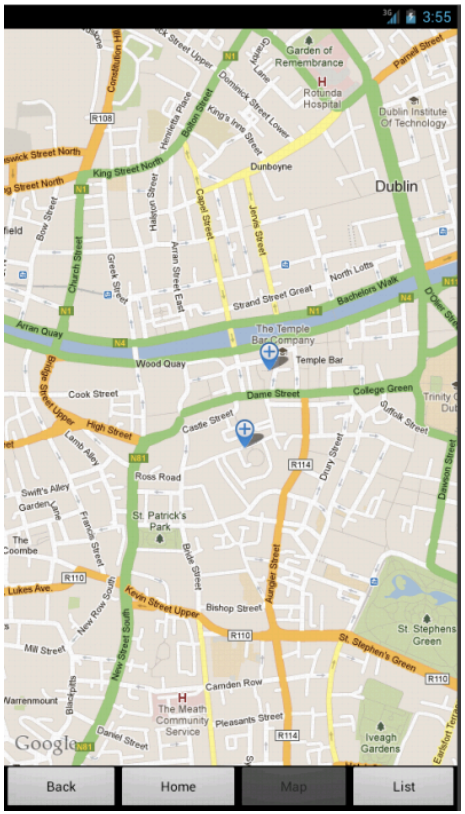


Figure 6-2: First Implementation with Same pins for all Restaurants

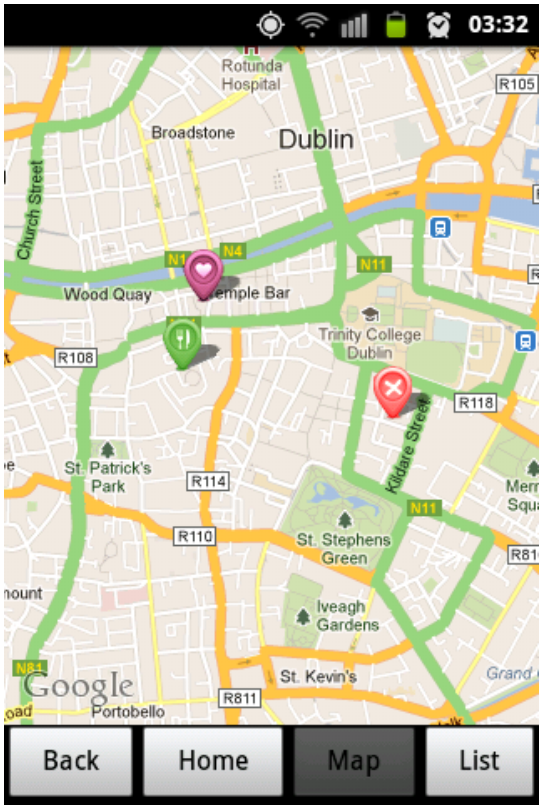


Figure 6-3: Final Screen with coloured pins for good/bad rated restaurants and users favourites

6.3 Web Database vs Local Database

Early in the project I decided that the content of the app should be dynamic. This made a central database that was accessible to all users the obvious option. The user will then retrieve content from this database each time they use the app.

Each time a user opens an Activity the app makes a call to the web database to get content, which is not stored in the phone but only visible for that instance of the Activity. This means a lot of calls to the server to get data even if the data has not changed between calls. While improvements in connectivity such as 3G, 4G and broadband Wi-Fi make the relative cost of this quite low in terms of battery life and network charges, it is still not as efficient a system as it could be

If I were to build the app again I would use a combination of Local and Web Databases. When the app first starts, I would retrieve all relevant data from the server and store it locally. Then only after a period of time (say 15 minutes) or when the user requested would a new call for updated data be made. When the user moves between activities they would access data from the local SQLite database, reducing the number of calls to the Web database dramatically. This option would require a lot more local storage but would save the user money and battery life.

6.4 Conclusion

The overall plan for this app did not change very much from the original intentions but it did increase in scope. Features were added to the plan and some of the core features were altered slightly in their interface. I achieved what I set out to build. The final app is true to the plan with added features to improve usability

7 Conclusion

7.1 Project Challenges

As I am new to software development I approached many challenges while building this app. Discounting learning Java and familiarising myself with android development there were a few key problems I struggled with. Most of them occurred while attempting to integrate the web database into the system.

I discovered that http requests cannot be made from an android app on the main user interface thread. I had to learn how to make asynchronous threads one of which is used to make calls to the web database.

Another headache was attempting to debug the PHP scripts. I did not find a tool to step through PHP scripts to find errors. Instead I used trial and error, printing out variables to the browser at various points to see if they were as expected.

A significant problem I encountered is that the entire app is trusting on the reliability of the hosting service. If the service in interrupted then the app will not work. As I am using a free service it is less reliable than paid services with little or no support. Once or twice during development I did encounter loss of service. This was very frustration as it was unknown when/if service would resume. If I were to launch the app I would move the website to a reliable and well trustworthy web hosting service.

7.2 Future Development

As discussed in section 6.3 I would explore the pros and cons of integrating local storage into the app. This would store the app content for the duration of a session.

I could also add a feature with “Our Recommendations”. These recommendations would be found by finding the highest rated restaurants and the restaurants that have the highest number of favourites by users. This would help users find the ‘best’ restaurants.

At moment all the restaurants in the database are shown to the users. However a user may have a few restaurants they strongly dislike. It may be useful to have the option

for the user to ‘blacklist’ restaurants. This would mean that those restaurants would no longer be shown to just them. Should the wish to give the restaurants a second change they should be able to ‘whitelist’ them again.

Another useful feature would allow the user to tap on a restaurant phone number or email and have the appropriate android activity open. This way a user could easily contact a restaurant to make a reservation. The phone call option would be implemented by specifying the app has the permission to make a phone call in the manifest file.

```
<uses-permission android:name=”android.permission.CALL_PHONE”>
        </uses-permission>
```

Then in the activity you can create an intention that gets called when a user taps the number.

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse(“tel:123456789”));
startActivity(callIntent);
```

I would also look at the possibility of monetising the app through premium listings for restaurants. Restaurants would pay to have a premium listing. A premium position would be displayed in the list view at the top of the list with bigger font and an image. The regular listings would not have an image. In the restaurant view the premium position would have larger icons. Of course in order for restaurants to want to buy these listing there would need to be a large number of users as potential customers.

7.3 User Acceptance

The content for this app is user generated which means that the more people using the app the better the content. However in order to get people to use the app the content must be good. This creates a kind of catch-22. Ideally speaking there should be sufficient content before the launch of the app so as not to frustrate early users. As the potential user groups of the app is quite small already it is important to retain as many users as possible. My plan to launch would go as follows:

1. Approach the various websites dedicated to food allergies to get initial users and testers. For example dublinwithfoodallergies.blogspot.ie. Also users could be found in forums such as boards.ie.
2. These users would help to add a number of restaurants to the database as well as reviews for these restaurants.
3. When a sufficient amount of content is in the database launch the app for free on the Google Play Store. This would be for a limited time.
4. After a period, when the free users have added enough data to make the content comprehensive I would then re-launch the app for a fee to be determined.

7.4 Summary

All in all I am very satisfied with my experience in creating this app. I implemented all the features I intended too as well as some additional ones. It was brand new experience for me that brought with it a lot of challenges and triumphs. I did not come to the project with many of the skills necessary to complete it. Coming out the other end my skill set is vastly expanded and my education is richer. I can now approach my next project with greater confidence, better planning and know that I can tackle any problem that arises.

8 Bibliography

8.1 Works Cited

[1] Survival Guide: Dublin with Food Allergies. [Online]. http://dublinwithfoodallergies.blogspot.ie/

[2] Subbu Allamaraju. (2008, December) InfoQ. [Online]. http://www.infoq.com/articles/subbu-allamaraju-rest

[3] Android. (2013, April) Android Developers. [Online]. http://developer.android.com/reference/android/test/ActivityInstrumentationTestCase2.html

[4] GSMArena. (2013, Febraruy) GSMArena. [Online]. http://www.gsmarena.com/state_of_android_jan_2013_jb_market_share_still_under_15-news-5489.php

[5] Google. (2013, March) Google Maps Android API v2. [Online]. https://developers.google.com/maps/documentation/android/start#the_google_maps_api_key

[6] I Answer 4 U. Spiral Model : Advantages and Disadvantages. [Online]. http://www.ianswer4u.com/2011/12/spiral-model-advantages-and.html#axzz2Sgwcvjf8

[8] Just In Mind. (2013, January) Justinmind. [Online]. http://www.justinmind.com/prototyper/free-edition

[7] Margaret Rouse. (2007, March) spiral model (spiral lifecycle model). [Online]. http://searchsoftwarequality.techtarget.com/definition/spiral-model

[9] Surinder Rajpal. (2012, December) Android Lifecycle. [Online]. http://programmingworld4u.blogspot.ie/

[10] Android App Market. (2012, Mar) Android App Market. [Online]. http://www.android-

app-market.com/android-activity-lifecycle.html

[11] Google Developers. (2013, April) Googel Developers. [Online]. https://developers.google.com/maps/documentation/android/v1/hello-mapview

[12] Robotium. (2013, April) Robotium. [Online]. https://code.google.com/p/robotium/wiki/Getting_Started

[13] Corey Goldberg. Pylot - Getting Started Guide. [Online]. http://www.pylot.org/gettingstarted.html

8.2 Images

8.3 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.html

Figure 3-3 - Spiral Methodology

Retrieved April 29, 2013, from Qmetry website:

http://www.qmetry.com/spiral.html

Figure 4-2 – Android Activity Lifecycle

Retrieved May 2, 2013, from Programming World website:

http://programmingworld4u.blogspot.ie/

9 Appendix

9.1 Initial System Architecture Diagram

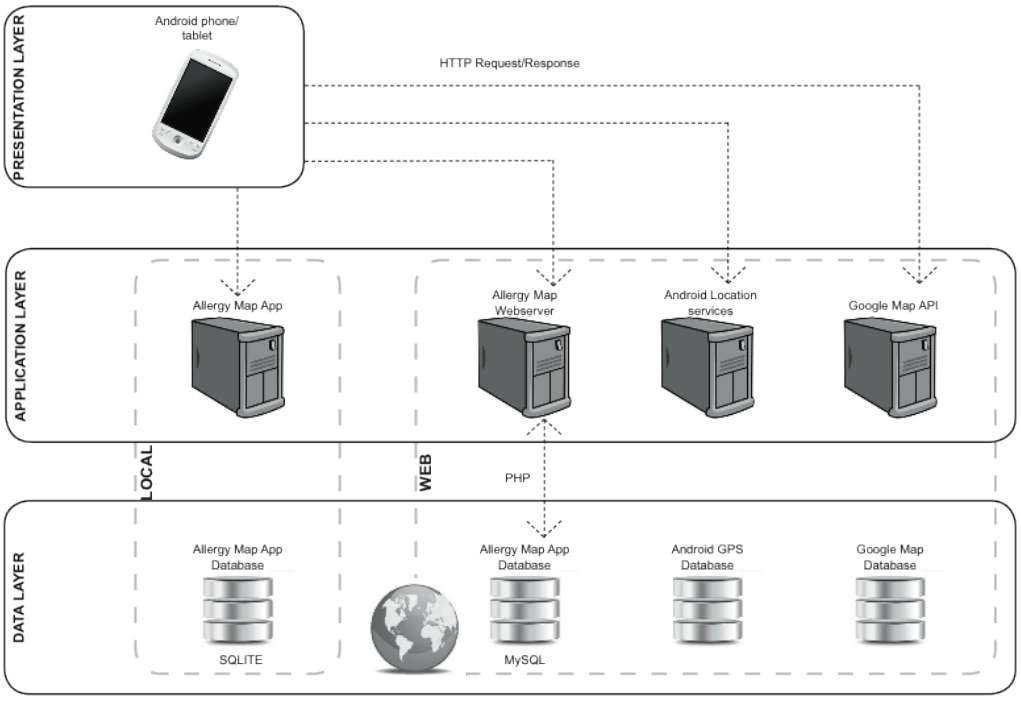


Figure 9-1: Initial System Architecture Design

9.2 Prototype Screens

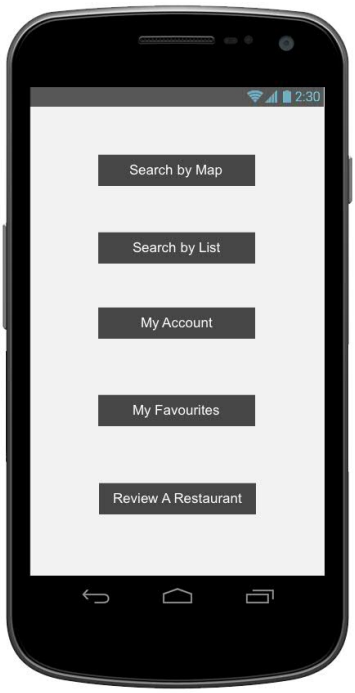


Figure 9-2: Home Screen

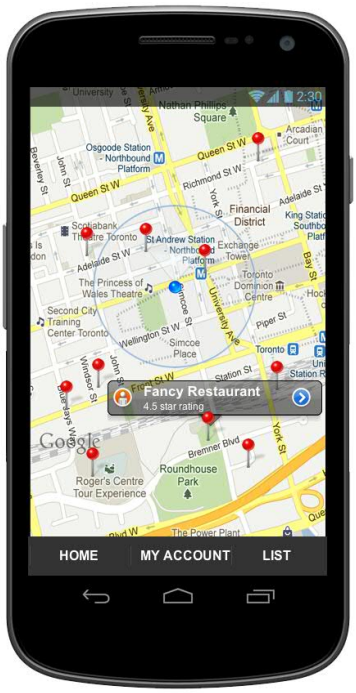


Figure 9-3: Map Screen

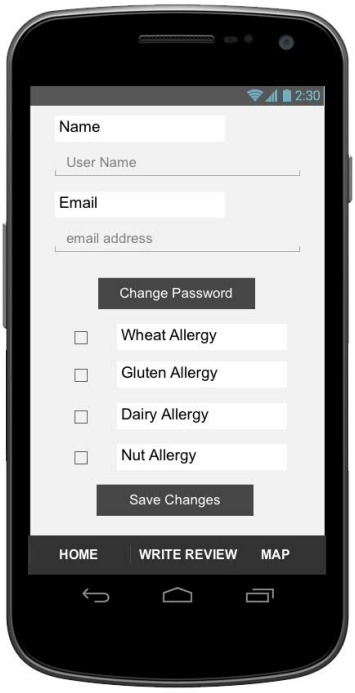


Figure 9-4: My Account

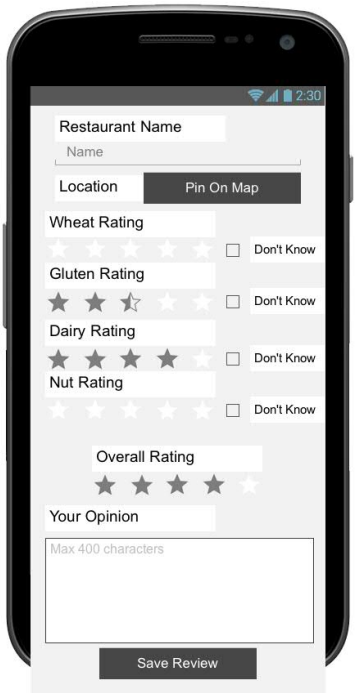


Figure 9-5: Review Restaurant

9.3 First Implementation of Screens

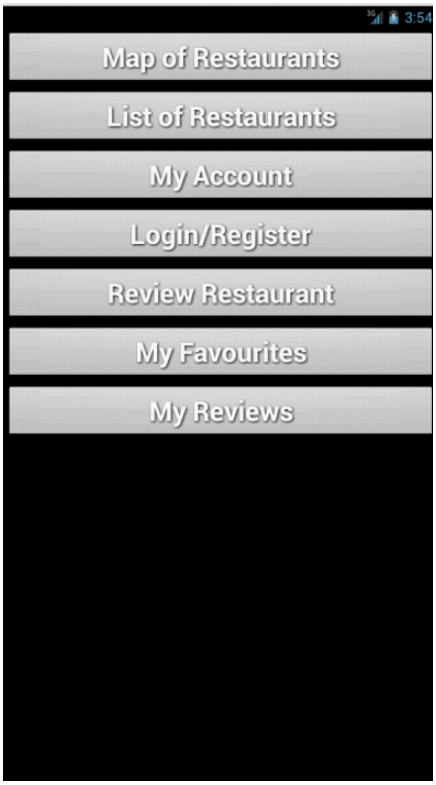


Figure 9-6: Main Menu

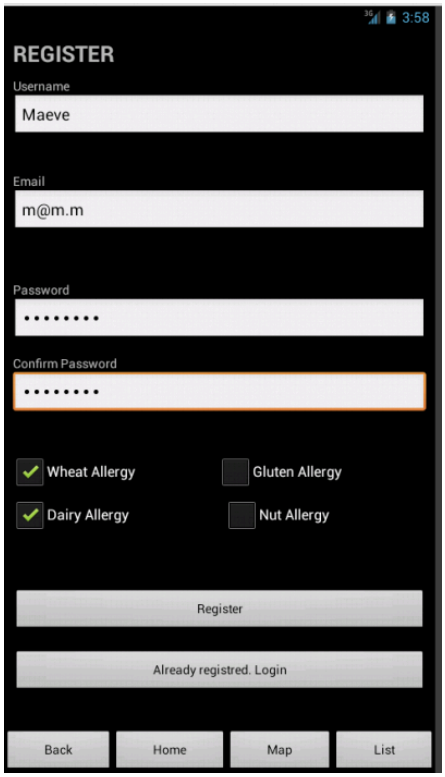


Figure 9-7: Register Account

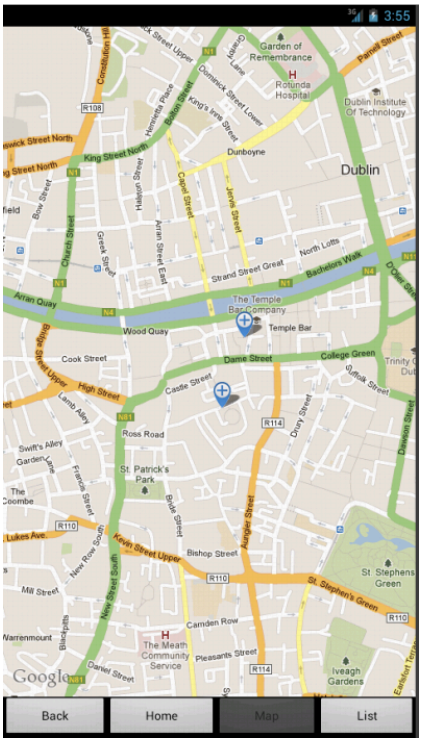


Figure 9-8: Map View

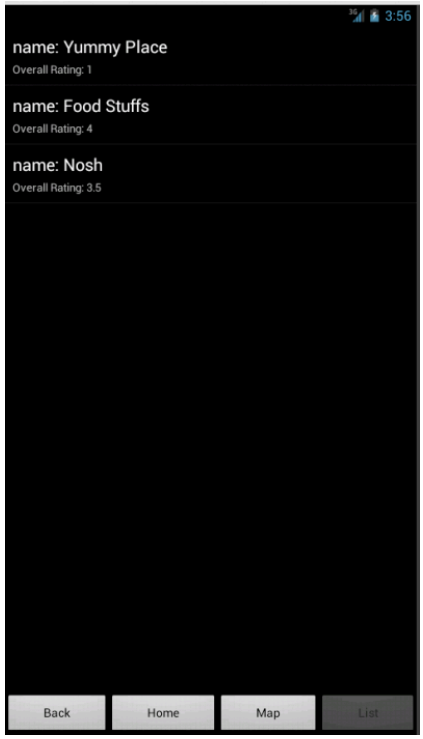


Figure 9-9: List View of restaurants

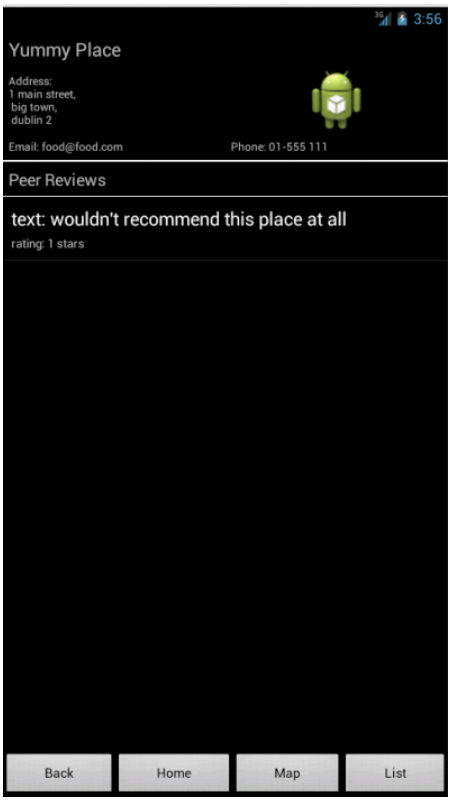


Figure 9-10: Restaurant Detail

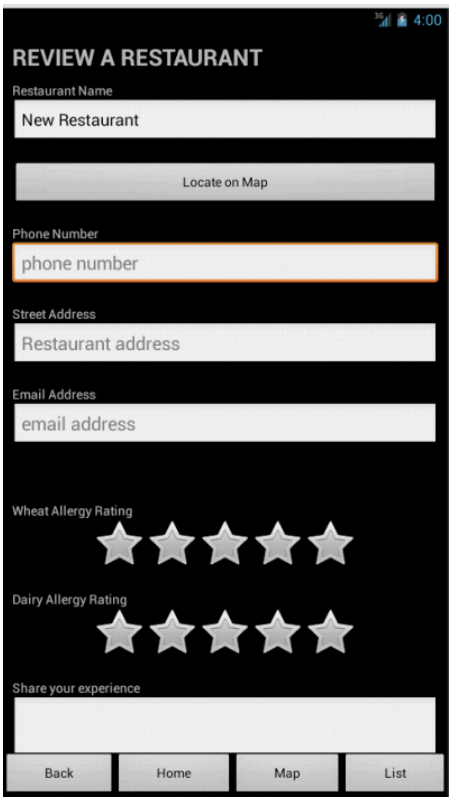


Figure 9-11: Review Restaurant