Bingo Deals

Project Report

Comprehensive Report of Bingo Deals application including Introduction, literature review, system design-diagrams& justifications, important implementation details and the conclusion.

Shiluka Raveen Dharmasena **Index: 110123N**

19-Sep-14

Table of Contents

Introduction

People are always looking for great deals around them wherever they stay or traveling, but still there are no flexible solutions to meet their demands except traditional method such as email marketing, social media ads etc. It is difficult and irritating to check emails for deals around them, but if we can give updates automatically about deals to our smart phones wherever we travel, it is really awesome.

Bingo is an innovative smart phone app uses android with GPS technology which helps user to get daily updates such as deals/discounts from hotels, restaurants, cafes, movie theatres, super markets or virtually anything near according to his/her current location

There is a web site associate with Bingo application. First user should register for a user account with his/her email address and mobile phone no. Then user need to select whatever he/she wants to receive, such as deals or coupons (Hotel Deals, electronics, Food like mcDonelds, KFC etc).

After that user can install android app to his/her smart phone and login with credentials from site registration. Then wherever they go, according to location they will receive nearest deal alerts via app/email or SMS according to ability of the function. Main benefit to user is, he/she can get deals according to geo location without any effort.

Main objective of this app is to send deal alerts when the locations are discovered. If mobile has GPS, deals can be sent as mails (notification popup). Addition to that it finds user's current location plotted on a map and displays details of nearby discovered sites such as address, phone number, distance from current location, ratings, reviews etc.

Literature Review

Project Schedule

Category	Sub Category	Description	Time Period
Research into project and the tools available	Research about the Google map API	Permissions to get from the google api console, Research about JSON parser, Place JSON parser, DirectionsJSON parsers. Research about google nearby places and google directions api.	June 1st – June 14th
	Research about push notification sending from server	Pick GCM (Google Cloud Message) server to send push notifications.	June 14th - June 21st
	Research about database management at the server	GCM (google cloud message) server, SQLite database (inside android os), SQLiteOpenHelper.	June 21st – June 28th
	Research about android system	Android alert manager, android wake locker	June 28th – July 04th
	Research about locations tracker	How to get current location, get latitude and longitude using gps or from the network.	June 04th – July 10th
Designing the app	Design the user's location identification system	Get the current Location of the user. Using gps or the network available.	July 10th – July 15th
	Design the nearby sites discovering system	Get the nearest places of the user using google map api.	July 15th – July 20th
	Design the database	Designing the GCM user database, Sites Database (to keep static deals), and the SQLite database to keep the live messages.	July 20th – July 25th
	Design the	Use GCM to send deals to users via (push	July 25th – Aug

	deals/discounts sending system	notifications)	2nd
Implementation and Testing of the app	Implement the user's location identification and test	Implement the GPS tracker, get the permission from google api. Unit testing the system.	Aug 2nd – Aug 10th
	Implement the nearby sites discovering system and test	Get the nearby places by sending http request and use google nearby places request to get nearby places of a given location. Unit test the sub system.	Aug 10th – Aug 16th
	Implement the get direction to the given location system and test	Implement JSONDirection parser	Aug 16th – Aug 25th
	Implement the server, sites, and the live message database systems and test	Implement the GCM user database, Site database to keep deals, SQLite database to keep live messages and test	Aug 25th – Sep 1st
	Implement the deals/discounts sending system (push notification system) and test	Implement GCM push notification system, modify GCMBaseIntentService.	Sep 1st – Sep 6th
	Testing of the final product	Test the final product as integrating testing. Connect the entire unit test together and testing.	Sep 6th – Sep 17th

Description

People are always looking for great deals around them wherever they stay or traveling, but still there are no flexible solutions to meet their demands except traditional method such as email marketing, social media ads etc. It is difficult and irritating to check emails for deals around them, but if we can give updates automatically about deals to our smart phones wherever we travel, it is really awesome. There is a website associate with Bingo application. First user should register for an account with email address and mobile phone no. Then need to select whatever wants to receive, such as deals or coupons. After that user can install android app to smart phone and login with credentials from site registration. Then wherever user goes, according to location will receive nearest deal alerts via app.

Application Functions

Application has a capable of providing accurate responses quickly and Interfaces of the Bingo application and the websites are user friendly.

Application Constraints

- GUI and website in English language
- Only registered users will be authorized to use the service of live messaging
- Users should install Bingo app on their android devices
- Android device should have GPS technology

Assumptions

- Users are familiar with the functionalities of the Bingo application
- Android device should have GPS technology

Functionality

Register for an account from the application

- User can sign up for an account from the *application*. User need to *only enter the email address*

Login to the account

- User can login to their account using username and password

Select whatever user wants to receive from the site

- There are several types of deals or coupons available such as Hotels, Super markets and Cafes. So the user can select what kind of deals he/she wants from the website.

Receive deals

- User can receive deals from nearby locations by clicking send deals button in the Bingo application.

Get the nearby locations

- User can get the information about nearby locations such as distance, reviews, address, and mobile number from the Bingo application.

Usability

Required training time

- Training time for a normal user is about 2 hrs and training time for a power user about 20 minutes. Application gives error messages and how to recover from error when erro occurs. Server which website is hosted is available at least 98% since users need to register via the site. Mean time is between failures are more than 5 months. User can get accurate information about nearby locations and accurate deals from those sites. Accuracy is more than 90%

Performance

- Response time in website
- Response time in website is less than 1 second. User should be able to register, login and select types from site as quickly as possible

- Response time in Bingo application
- Response time in Bingo application is less than 5 seconds. Deals are sent to the application within 5 seconds

Design Constraints

Android

- Platform of the application is Android

Java

- Application uses Java as the default programming language

Google Map API

- Location identification is done using Google Map API

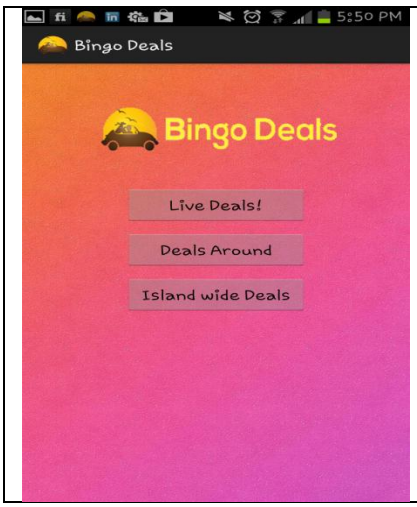
Eclipse

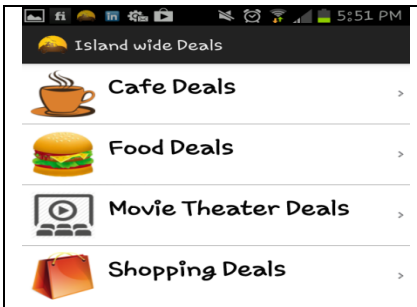
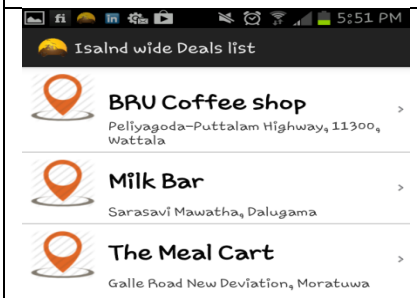
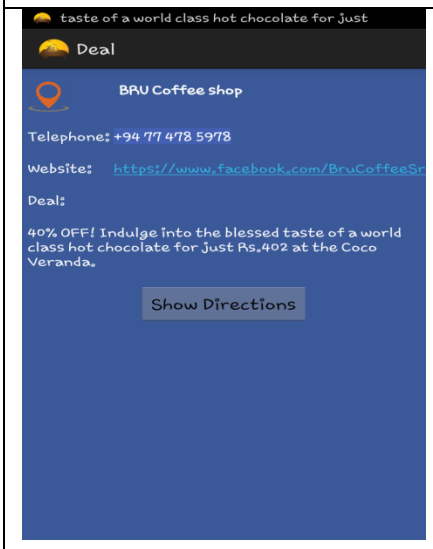
- Eclipse is used as IDE

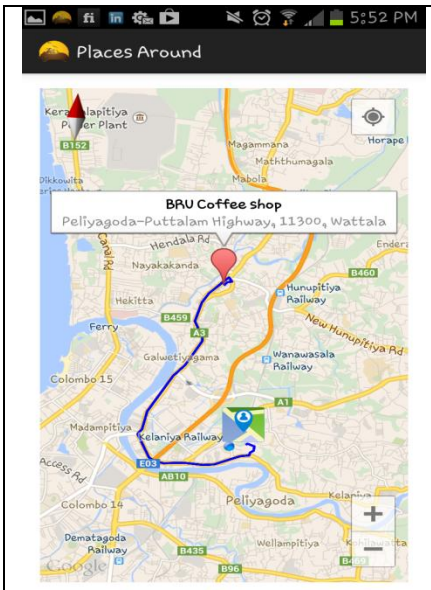
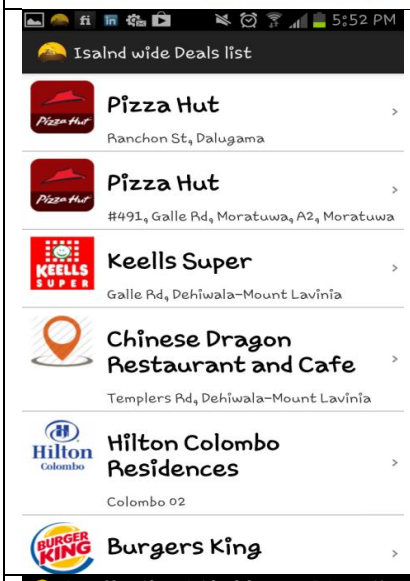
Interfaces

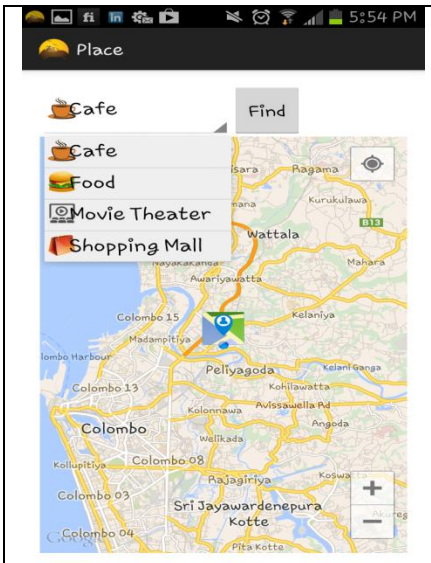
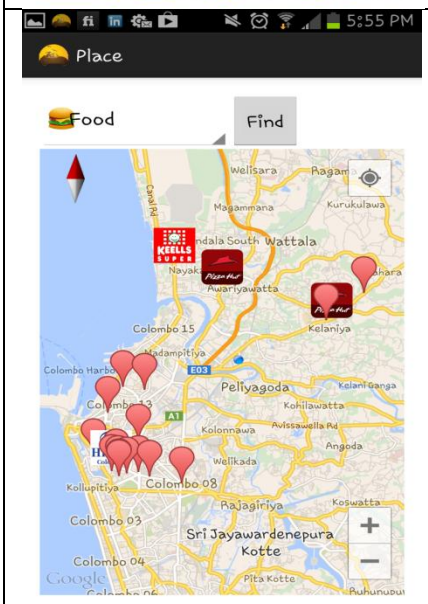
User Interfaces

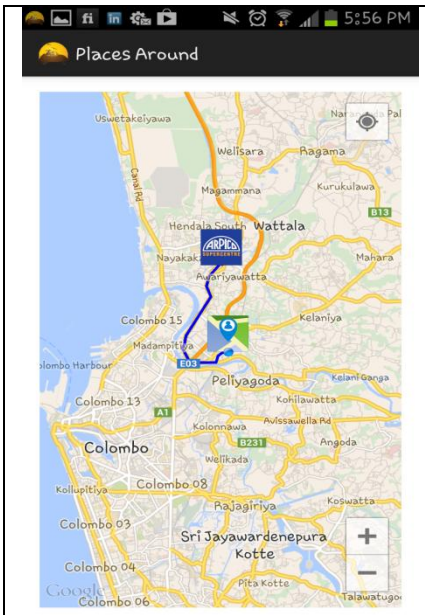
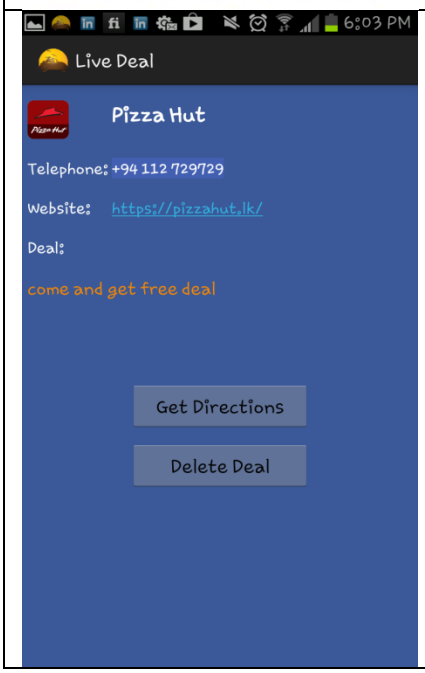
- User interface of the bingo application is easy to use.

		Main Window of the Application Live Deals – show the Live Messages from the site Deals Around – show the Deals around your current location Island wide Deals – show all the deals

	Island wide Deals window User can choose which kind of category he/she needs. There are 4 types of categories 1. Café Deals 2. Food Deals 3. Movie Theater Deals 4. Shopping Deals
	Island wide Deals List Show all the deals from database (Name of the location and the vicinity) Currently show the default icon of the location
	Deal Window There are several details about the deal Location Name – which location gives the deal Telephone number of the location Website of the location Deal of the Location Direction to the location can be viewed by clicking the “Show Directions” Button

	Show the Place using Google maps Clicking the marker can get information of the place Route to the place is shown by the purple line
	Deals Category List Show the Place Name and the Vicinity Icon shows the identity of the place
	Deal Example There can be shown maximum 3 deals. Clicking “Show Directions” Button user can get the location of the place

	Nearby Places Categorization Café Food Movie Theatre Shopping Mall User can select which type of information he/she needs
	Showing nearby places on Google Map If the places are located in the Database they are shown by their Icons

	Identifying route for the given Location
	Live Deal Example Places can send Live Deals to their customers. So the customers can get the directions and visit the site

Hardware Interfaces

- Minimum Requirements
    - Smart phone with android OS higher than version 2.2 (Froyo)
    - Smart phone with GPS technology

Software Interfaces

- Google Map API - Google Map API is used for identify the location and the nearby sites to the user.
    - Web Server

      To host the website associated with Bingo Application.
    - Database Server

      There are mainly two databases. First one keeps records of the user and the second one keeps records of the sites.
    - Development End

      Java, mysql, php, android (OS)

Communications Interfaces

- User's location can be identified by using GPS technology
    - Message sending happens using internet (HTTP/HTTPS) protocol

System design-diagrams

Architectural Representation

- **Logical view**

  Actor: Designers.

  Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations.

  Related tools: Design model

- **Process view**

  Actor: Developers

  Area: Non-functional requirements: describes the design's concurrency and synchronization aspects.

- **Implementation view**

  Actor: Programmers.

  Area: Software components: describes the layers and subsystems of the application.

  Related tools: Implementation model, components

- **Deployment view**

  Actor: Deployment managers.

  Area: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects.

- **Use Case view**

Actor: all the stakeholders of the system, including the end-users.

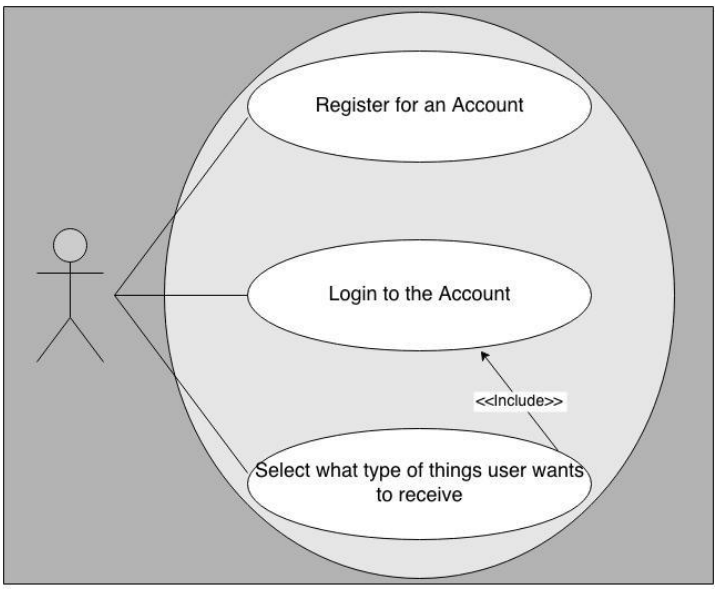
Area: describes the set of scenarios, use cases that represent some significant, central functionality of the system.

Related tools: Use-Case Model

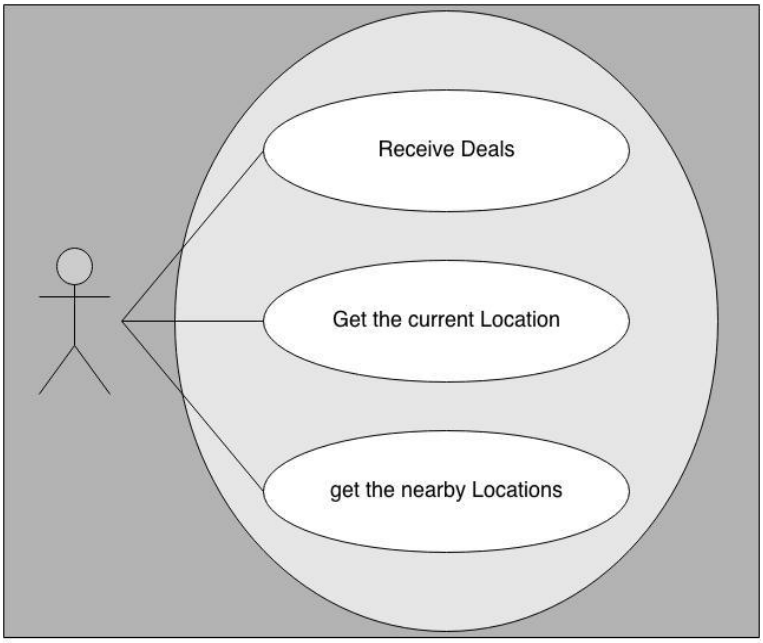
Use-Case View

-
    - Register for an account from the website
- User can sign up for an account from the website. User need to enter username, password and mobile phone number
    - Login to the account
- User can login to their account using username and password
    - Select whatever user wants to receive from the site
- There are several types of deals or coupons available such as Hotels, Super markets and Cafes. So the user can select what kind of deals he/she wants from the website.
    - Receive deals
- User can receive deals from nearby locations by clicking send deals button in the Bingo application.
    - Get the nearby locations
- User can get the information about nearby locations such as distance, reviews, address, and mobile number from the Bingo application.

Website



Application



Use-Case Realizations

- .

Use Case	Register for and account	
Description	When user visits to the website, website asks to sign up if user hasn't an account.	
Actors	User	
Preconditions	The user must visit to the site.	
Post conditions	To be sign up successful user must enter the user name, password, password again and the mobile number	
Flow of events	Actor	System
	1. User visits to the website	1.1 show sign up button
	2. User register for an account	2.1 show sign up window
Exception conditions	If the sign up information wrong, website should give a massage saying what was the wrong	

Use Case	Login to Website	
Description	When a user login to the website, website asks to enter student user name and the password.	
Actors	User	
Preconditions	The user must be registered in website.	
Post conditions	To be login successful user must enter the user name and password correctly.	
Flow of events	Actor	System
	1 User visits to the website	1.1 show login window
	2 User login to the website	2.1 authenticate user
Exception conditions	If the user name or the password wrong, website should give a massage saying login is incorrect.	

Use Case	Select what type of deals user wants to receive
Description	After user login to the website, website asks what type of deals user wants to get. (Hotel deals, restaurant deals, super market deals)
Actors	User
Preconditions	The user must login to the site.

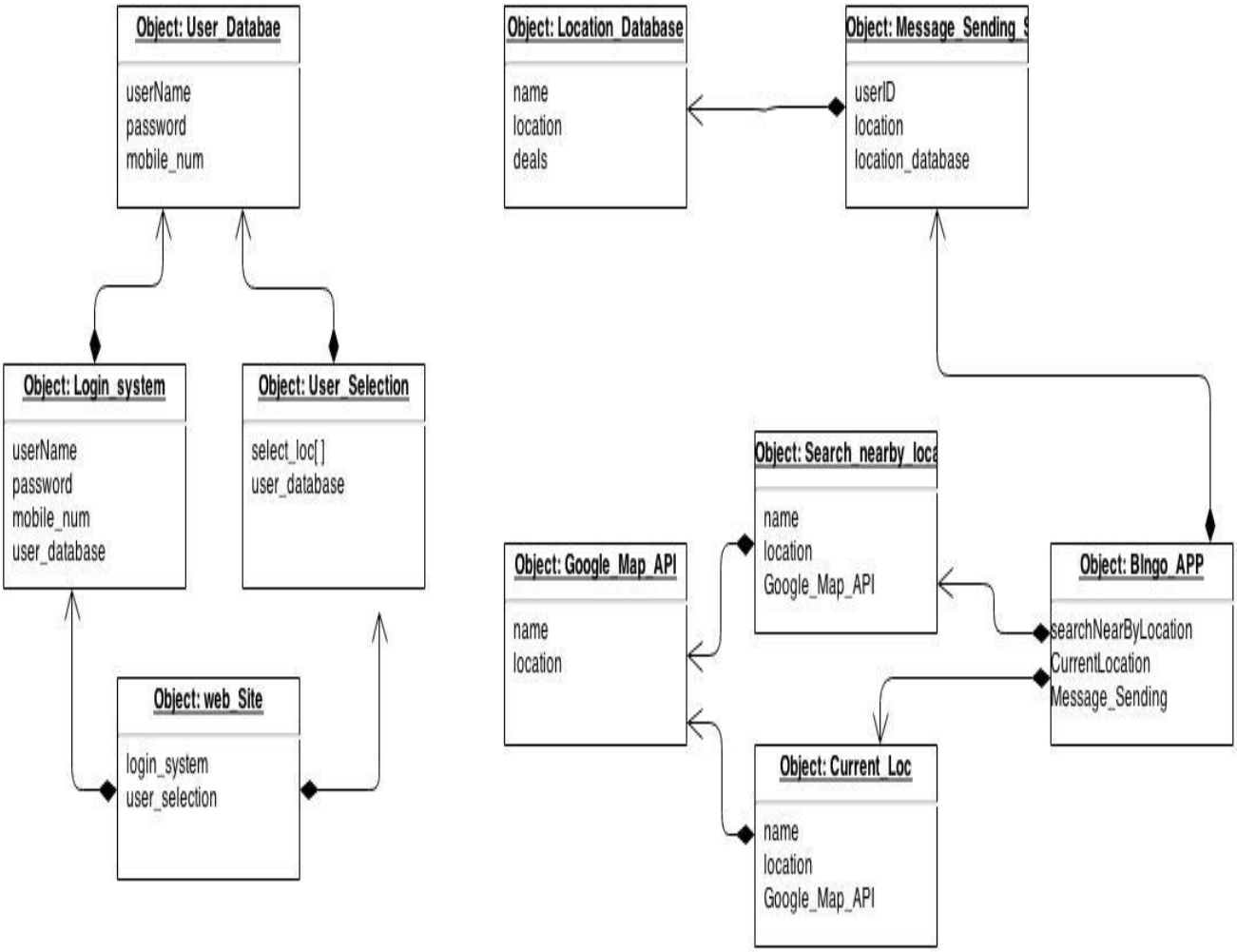
Post conditions	By default all the options are available	
Flow of events	**Actor**	**System**
	1 User login to the website	1.1 show selection window
	2 Select what kind of deals wants	2.1 show selection window
Exception conditions	User should select at least one type, otherwise website should give a error massage	

Use Case	**Receive Deals**	
Description	Application should received deals from the nearby locations when user clicks receive deals button	
Actors	User	
Preconditions	The user must run Bingo Android application	
Post conditions		
Flow of events	**Actor**	**System**
	1 Run the Bingo application	1.1 show 'send deals' button
	2 click 'send deals' button	2.1 send deals from nearby locations
Exception conditions		

Use Case	**Show the current location**	
Description	Application should the current location using GPS or IP	
Actors	User	
Preconditions	The user must run Bingo Android application	
Post conditions		
Flow of events	**Actor**	**System**
	1 Run the Bingo application	1.1 show 'current location' button
	2 click 'current location button	2.1 show the current location
Exception conditions		

Use Case	Get nearby locations	
Description	Application should show nearby locations of the user	
Actors	User	
Preconditions	The user must run Bingo Android application	
Post conditions		
Flow of events	**Actor**	**System**
	1 Run the Bingo application	1.1 show nearby locaitons
Exception conditions		

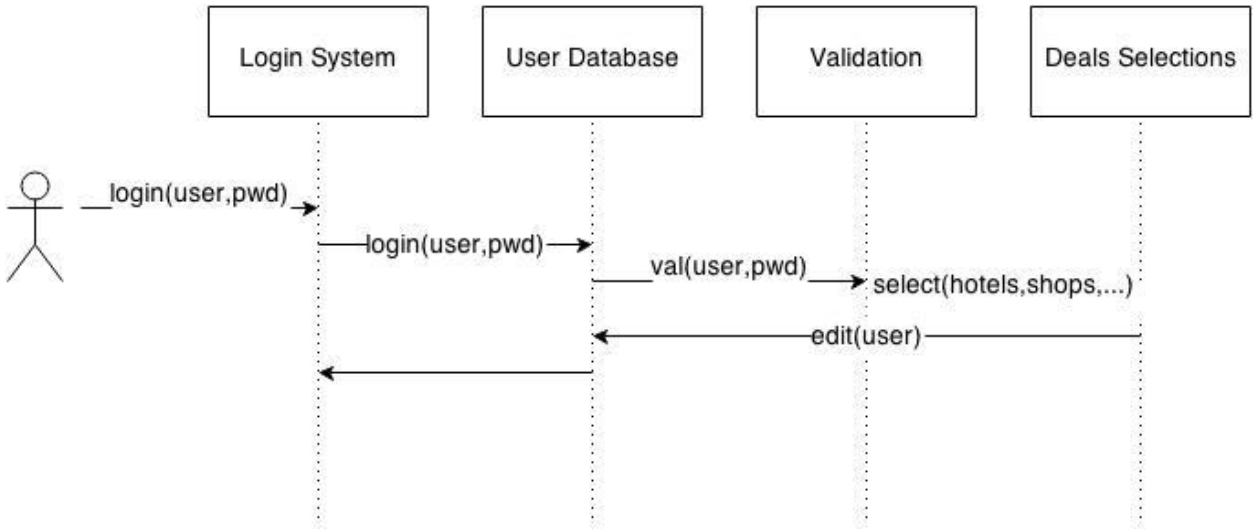
Logical View



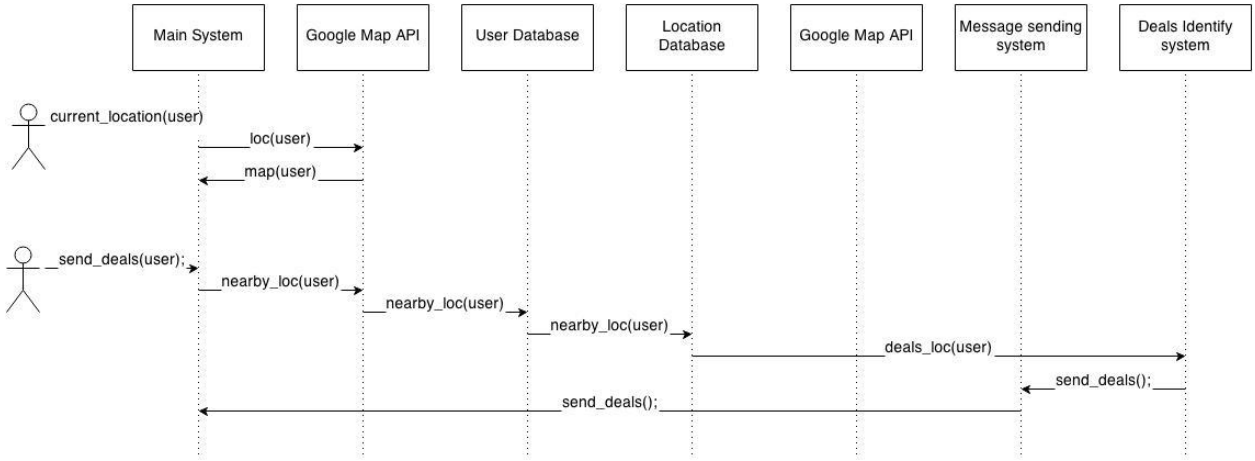
Process View

-

Process of Login System

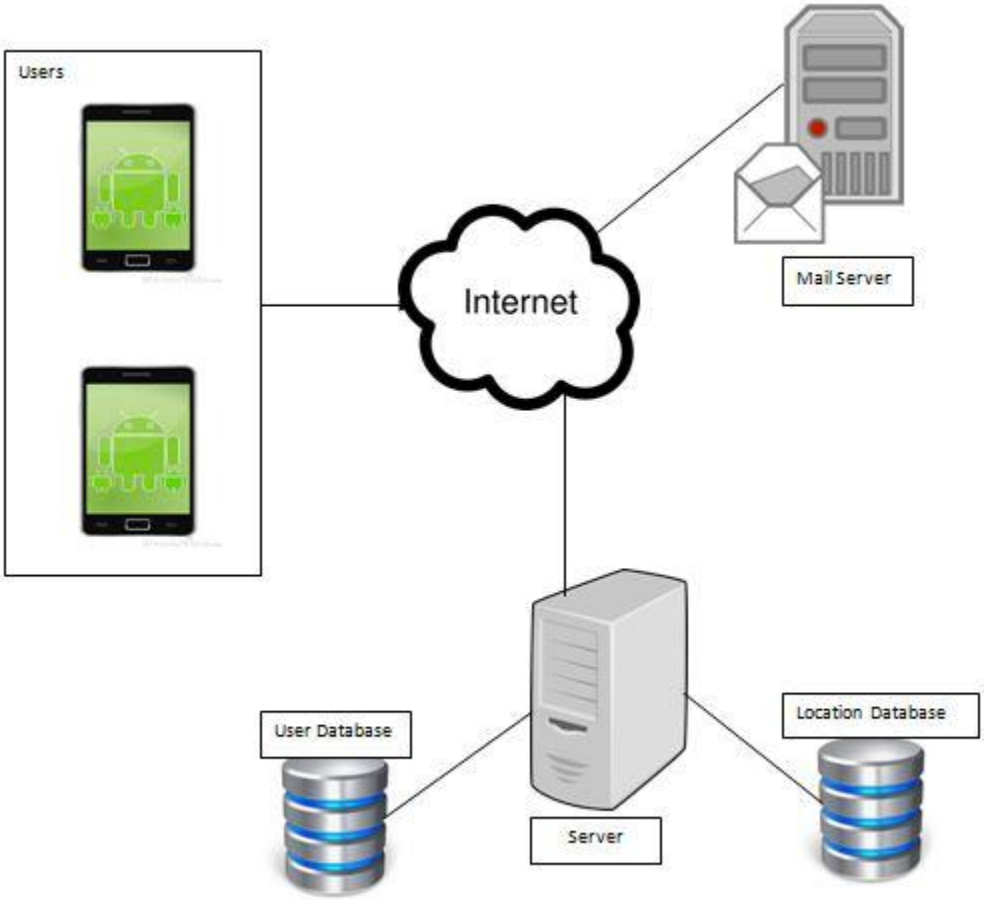


Process of Bingo Application



Deployment View

- Deployment View of the Bingo Application



Important implementation details

JSON Parser Class

```
package com.softzone.parser;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;


/**
 * JSON Parser for get http requests
 */

public class JSONParser {

	static InputStream is = null;
	static JSONObject jObj = null;
	static String json = "";

	// constructor
	public JSONParser() {

	}

	// function get json from url
	// by making HTTP POST or GET mehtod
	public JSONObject makeHttpRequest(String url, String method,
		List<NameValuePair> params) {
```

```
		// Making HTTP request
		try {

			// check for request method
			if (method.equalsIgnoreCase("POST")) {
				// request method is POST
				// defaultHttpClient
				DefaultHttpClient httpClient = new DefaultHttpClient();
				HttpPost httpPost = new HttpPost(url);
				httpPost.setEntity(new UrlEncodedFormEntity(params));

				HttpResponse httpResponse = httpClient.execute(httpPost);
				HttpEntity httpEntity = httpResponse.getEntity();
				is = httpEntity.getContent();

			} else if (method.equalsIgnoreCase("GET")) {
				// request method is GET
				DefaultHttpClient httpClient = new DefaultHttpClient();
				String paramString = URLEncodedUtils.format(params, "utf-
8");
				url += "?" + paramString;
				HttpGet httpGet = new HttpGet(url);

				HttpResponse httpResponse = httpClient.execute(httpGet);
				HttpEntity httpEntity = httpResponse.getEntity();
				is = httpEntity.getContent();
			}

		} catch (UnsupportedEncodingException e) {
			e.printStackTrace();
		} catch (ClientProtocolException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}

		try {
			BufferedReader reader = new BufferedReader(new InputStreamReader(
					is, "iso-8859-1"), 8);
			StringBuilder sb = new StringBuilder();
			String line = null;
			while ((line = reader.readLine()) != null) {
				sb.append(line + "\n");
			}
			is.close();
			json = sb.toString();
		} catch (Exception e) {
			Log.e("Buffer Error", "Error converting result " + e.toString());
		}

		// try parse the string to a JSON object
		try {
			jObj = new JSONObject(json);
		} catch (JSONException e) {
			Log.e("JSON Parser", "Error parsing data " + e.toString());
```

```
            }

            // return JSON String
            return jObj;

        }
}
```

Decoding Polylines from Google Maps Direction API with Java

http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java

```
private List<GeoPoint> decodePoly(String encoded) {

        List<GeoPoint> poly = new ArrayList<GeoPoint>();
        int index = 0, len = encoded.length();
        int lat = 0, lng = 0;

        while (index < len) {
                int b, shift = 0, result = 0;
                do {
                        b = encoded.charAt(index++) - 63;
                        result |= (b & 0x1f) << shift;
                        shift += 5;
                } while (b >= 0x20);
                int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >>
1));
                lat += dlat;

                shift = 0;
                result = 0;
                do {
                        b = encoded.charAt(index++) - 63;
                        result |= (b & 0x1f) << shift;
                        shift += 5;
                } while (b >= 0x20);
                int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >>
1));
                lng += dlng;

                GeoPoint p = new GeoPoint((int) (((double) lat / 1E5) * 1E6),
                         (int) (((double) lng / 1E5) * 1E6));
                poly.add(p);
        }

        return poly;
}
```

Getting nearby deals

```
protected void onPostExecute(String file_url) {
                    // dismiss the dialog after getting all sites
                    pDialog.dismiss();
                    // updating UI from Background Thread
                    runOnUiThread(new Runnable() {
                          public void run() {

                                try {

                                      // sleep for 3 seconds while downloading data
                                      Thread.sleep(3000);

                                      // get the sitesList
                                      // if sites were not in
NearbyPlacesActivity.places
                                      // remove the site

                                      for (int i = 0; i < sitesList.size(); i++) {
                                            int x = 0;
                                            for (int j = 0; j <
NearbyPlacesActivity.places
                                                        .size(); j++) {

                                                  if (sitesList
                                                              .get(i)
                                                              .get("gid")

      .equals(NearbyPlacesActivity.places

      .get(j).get("place_id"))) {

                                                        x++;

                                                  }
                                            }
                                            if (x == 0) {
                                                  sitesList.remove(i);
                                                  i--;
                                            }
                                      }

                                } catch (InterruptedException e) {
                                      // TODO Auto-generated catch block
                                      e.printStackTrace();
                                }

                                /**
                                 * Updating parsed JSON data into ListView
                                 * */
                                ListAdapter adapter = new SimpleAdapter(
                                            NearSitesActivity.this, sitesList,
```

```
                                                R.layout.list_site_near, new String[] {
TAG_LID,
                                                            TAG_NAME, TAG_VICINITY,
TAG_IMAGE },
                                                new int[] { R.id.lid, R.id.name,
R.id.vicinity,
                                                            R.id.list_image });
                                    // updating listview
                                    setListAdapter(adapter);

                                }
                            });

                        }
```

Live Message Activity Class

```
package com.softzone.bingodeals.livemsg;

import static com.softzone.bingodeals.CommonUtilities.EXTRA_MESSAGE;
import static com.softzone.bingodeals.CommonUtilities.SENDER_ID;

import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.apache.http.NameValuePair;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.database.Cursor;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
```

```java
import android.widget.ImageView;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

import com.softzone.alert.AlertDialogManager;
import com.softzone.alert.WakeLocker;
import com.softzone.bingodeals.ConnectionDetector;
import com.softzone.bingodeals.ServerUtilities;
import com.google.android.gcm.GCMRegistrar;

import com.softzone.bingodeals.R;

import com.softzone.sqlitedb.DatabaseHandler;
import com.softzone.sqlitedb.LiveMessage;

/**
 *
 * show live messages on list get the message from GCM
 */

public class LiveMsgActivity extends ListActivity {

	// Progress Dialog
	private ProgressDialog pDialog;

	// label to display gcm messages
	TextView lblMessage;

	// Asyntask
	AsyncTask<Void, Void, Void> mRegisterTask;

	// Alert dialog manager
	AlertDialogManager alert = new AlertDialogManager();

	// Connection detector
	ConnectionDetector cd;

	public static String email;

	private int msgId;
	private String msgName, msgGid, msgVicinity, msgMsg, msgImage;

	// list of messages in DB
	ArrayList<HashMap<String, String>> messagesList;

	ImageView iv;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.common_list);
```

```
        cd = new ConnectionDetector(getApplicationContext());

        // Check if Internet present
        if (!cd.isConnectingToInternet()) {
            // Internet Connection is not present
            alert.showAlertDialog(LiveMsgActivity.this,
                    "Internet Connection Error",
                    "Please connect to working Internet connection",
false);
            // stop executing code by return
            return;
        }

        // Getting name, email from intent
        Intent i = getIntent();

        email = i.getStringExtra("email");

        // Make sure the device has the proper dependencies.
        GCMRegistrar.checkDevice(this);

        // Make sure the manifest was properly set - comment out this line
        // while developing the app, then uncomment it when it's ready.
        GCMRegistrar.checkManifest(this);

        registerReceiver(mHandleMessageReceiver, new IntentFilter());

        // Get GCM registration id
        final String regId = GCMRegistrar.getRegistrationId(this);

        // Check if regid already presents
        if (regId.equals("")) {
            // Registration is not present, register now with GCM
            GCMRegistrar.register(this, SENDER_ID);
        } else {
            // Device is already registered on GCM
            if (GCMRegistrar.isRegisteredOnServer(this)) {
                // Skips registration.
                Toast.makeText(getApplicationContext(),
                        "Already registered with BingoDeals",
Toast.LENGTH_LONG)
                        .show();
            } else {
                // Try to register again, but not in the UI thread.
                // It's also necessary to cancel the thread onDestroy(),
                // hence the use of AsyncTask instead of a raw thread.
                final Context context = this;
                mRegisterTask = new AsyncTask<Void, Void, Void>() {

                    @Override
                    protected Void doInBackground(Void... params) {
                        // Register on our server
                        // On server creates a new user
```

```
                                        ServerUtilities.register(context, email,
regId);
                                        return null;
                                }

                                @Override
                                protected void onPostExecute(Void result) {
                                        mRegisterTask = null;
                                }

                        };
                        mRegisterTask.execute(null, null, null);
                }
        }

        /*
         * listing view
         */
        // Hashmap for ListView
        messagesList = new ArrayList<HashMap<String, String>>();

        // Loading sites in Background Thread
        new LoadAllSites().execute();

        // listview
        ListView lv = getListView();

        // select a single site and launch edit screen
        lv.setOnItemClickListener(new OnItemClickListener() {

                @Override
                public void onItemClick(AdapterView<?> parent, View view,
                                int position, long id2) {
                        // getting values from selected ListItem
                        String lid = ((TextView)
view.findViewById(R.id.id)).getText()
                                        .toString();

                        String liveMsg = ((TextView) view.findViewById(R.id.name))
                                        .getText().toString();

                        String msgDetail = ((TextView)
view.findViewById(R.id.msg))
                                        .getText().toString();

                        // Starting new intent (EditSiteActivity)
                        Intent in = new Intent(getApplicationContext(),
                                        LiveSitesActivity.class);
                        // sending lid to next activity
                        in.putExtra("name", liveMsg);

                        in.putExtra("msgDetail", msgDetail);

                        // starting new activity and expecting some response back
                        startActivityForResult(in, 100);
```

```
                }
            });

            // SQLite database for live messages
            DatabaseHandler db = new DatabaseHandler(this);

            List<LiveMessage> messages = db.getAllMessages();

            for (LiveMessage cn : messages) {
                String log = "Id: " + cn.getId() + " ,Name: " + cn.getName()
                            + " ,Vicinity: " + cn.getVicinity() + " ,GID: "
                            + cn.getGid() + " ,msg: " + cn.getMsg();

                msgId = cn.getId();
                msgGid = cn.getGid();
                msgName = cn.getName();
                msgVicinity = cn.getVicinity();
                msgMsg = cn.getMsg();

                // testing purposes
                Log.d("***Name: ", log);

            }

        }

        /**
         * Receiving push messages
         * */
        private final BroadcastReceiver mHandleMessageReceiver = new
BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                String newMessage = intent.getExtras().getString(EXTRA_MESSAGE);

                // Waking up mobile if it is sleeping
                WakeLocker.acquire(getApplicationContext());

                // Showing received message
                lblMessage.append(newMessage + "\n");
                Toast.makeText(getApplicationContext(),
                            "New Message: " + newMessage,
Toast.LENGTH_LONG).show();

                // Releasing wake lock
                WakeLocker.release();
            }
        };

        @Override
        protected void onDestroy() {
            if (mRegisterTask != null) {
                mRegisterTask.cancel(true);
            }
            try {
```

```
                unregisterReceiver(mHandleMessageReceiver);
                GCMRegistrar.onDestroy(this);
            } catch (Exception e) {
                Log.e("UnRegister Receiver Error", "> " + e.getMessage());
            }
            super.onDestroy();
        }

        /**
         * Background Async Task to Load all sites by making HTTP Request database
         * connection
         * */
        class LoadAllSites extends AsyncTask<String, String, String> {

            /**
             * Before starting background thread Show Progress Dialog
             * */
            @Override
            protected void onPreExecute() {
                super.onPreExecute();
                pDialog = new ProgressDialog(LiveMsgActivity.this);
                pDialog.setMessage("Loading Messages. Please wait...");
                pDialog.setIndeterminate(false);
                pDialog.setCancelable(false);
                pDialog.show();
            }

            /**
             * getting All sites from url
             * */
            protected String doInBackground(String... args) {
                // Building Parameters
                List<NameValuePair> params = new ArrayList<NameValuePair>();

                // getting message data from sqlite database
                DatabaseHandler db = new
DatabaseHandler(getApplicationContext());

                List<LiveMessage> messages = db.getAllMessages();

                for (LiveMessage cn : messages) {
                    String log = "Id: " + cn.getId() + " ,Name: " +
cn.getName()
                                + " ,Vicinity: " + cn.getVicinity() + " ,GID:
"
                                + cn.getGid() + " ,msg: " + cn.getMsg();

                    msgId = cn.getId();
                    msgGid = cn.getGid();
                    msgName = cn.getName();
                    msgVicinity = cn.getVicinity();
                    msgMsg = cn.getMsg();

                    String imageName = msgName;
```

```
                        // image - uri of the image
                        String msgImage = makeImageUri(imageName);

                        // creating new HashMap
                        HashMap<String, String> map = new HashMap<String,
String>();

                        map.put("msgId", Integer.toString(msgId));
                        map.put("msgGid", msgGid);
                        map.put("msgName", msgName);
                        map.put("msgVicinity", msgVicinity);
                        map.put("msgMsg", msgMsg);

                        map.put("msgImage", msgImage);

                        // adding HashList to ArrayList
                        messagesList.add(map);

                        // Writing Contacts to log
                        Log.d("*****Name: ", log);
                        System.out.println("****  " + log);
                    }

                    return null;
                }

                /**
                 * After completing background task Dismiss the progress dialog
                 * **/
                protected void onPostExecute(String file_url) {
                    // dismiss the dialog after getting all sites
                    pDialog.dismiss();
                    // updating UI from Background Thread
                    runOnUiThread(new Runnable() {
                        public void run() {
                            /**
                             * Updating parsed sqlite data into ListView
                             * */
                            ListAdapter adapter = new SimpleAdapter(
                                        LiveMsgActivity.this, messagesList,
                                        R.layout.list_live_msg, new String[] {
"msgId",
                                                "msgGid", "msgName",
"msgVicinity",
                                                "msgMsg", "msgImage" },
new int[] {
                                                R.id.id, R.id.gid,
R.id.name,
                                                R.id.vicinity, R.id.msg,
R.id.list_image });
                            // updating listview
                            setListAdapter(adapter);
                        }
                    });
```

```
            }

        }

        // make image uri to string
        private String makeImageUri(String name) {
            String image = name;
            // remove all white spaces
            String in = image.replaceAll("\\s+", "");
            // turn to lower case
            String iname = in.toLowerCase();
            System.out.println("iName is: " + iname);
            String mDrawableName = iname;
            // get the resId of the image
            int resID = getResources().getIdentifier(mDrawableName, "drawable",
                        getPackageName());

            // resID is notfound show default image
            if (resID == 0) {
                resID = getResources().getIdentifier("default_place", "drawable",
                            getPackageName());
            }

            // make the uri
            Uri imageURI = Uri.parse("android.resource://" + getPackageName() + "/"
                        + resID);
            image = imageURI.toString();
            return image;
        }

}
```

Place JSON Parser Class

```
package com.softzone.parser;


import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;


import org.json.JSONArray;
```

```
import org.json.JSONException;
import org.json.JSONObject;



/**
 * JSON Parser for get places
 */

public class PlaceJSONParser {

	/** Receives a JSONObject and returns a list */
	public List<HashMap<String, String>> parse(JSONObject jObject) {

		JSONArray jPlaces = null;
		try {
			/** Retrieves all the elements in the 'places' array */
			jPlaces = jObject.getJSONArray("results");
		} catch (JSONException e) {
			e.printStackTrace();
		}
		/**
		 * Invoking getPlaces with the array of json object where each json
		 * object represent a place
		 */
		return getPlaces(jPlaces);
```

```
}

private List<HashMap<String, String>> getPlaces(JSONArray jPlaces) {
	int placesCount = jPlaces.length();
	List<HashMap<String, String>> placesList = new ArrayList<HashMap<String, String>>();
	HashMap<String, String> place = null;

	/** Taking each place, parses and adds to list object */
	for (int i = 0; i < placesCount; i++) {
		try {
			/** Call getPlace with place JSON object to parse the place */
			place = getPlace((JSONObject) jPlaces.get(i));
			placesList.add(place);

		} catch (JSONException e) {
			e.printStackTrace();
		}
	}

	return placesList;
}

/** Parsing the Place JSON object */
private HashMap<String, String> getPlace(JSONObject jPlace) {
```

```
HashMap<String, String> place = new HashMap<String, String>();
String placeName = "-NA-";
String vicinity = "-NA-";
String place_id = "-NA-";
String latitude = "";
String longitude = "";

try {
        // Extracting Place name, if available
        if (!jPlace.isNull("name")) {
                placeName = jPlace.getString("name");
        }

        // Extracting Place Vicinity, if available
        if (!jPlace.isNull("vicinity")) {
                vicinity = jPlace.getString("vicinity");
        }

        if (!jPlace.isNull("place_id")) {
                place_id = jPlace.getString("place_id");
        }

        latitude = jPlace.getJSONObject("geometry")
                        .getJSONObject("location").getString("lat");
        longitude = jPlace.getJSONObject("geometry")
```

```
                                        .getJSONObject("location").getString("lng");


                        System.out.println("+++++" + placeName + "  " + vicinity
                                        + "  ++++++" + place_id + "+++++ " + latitude + " +++++ " +
longitude);


                        place.put("place_name", placeName);
                        place.put("vicinity", vicinity);
                        place.put("lat", latitude);
                        place.put("lng", longitude);


                        place.put("place_id", place_id);


                } catch (JSONException e) {
                        e.printStackTrace();
                }
                return place;
        }
}
```

Evaluation

Test Case	Description	status
Testing Alert Dialog Manager	Display alerts on mobile phone	✓
Testing wake locker	Acquire power services of the mobile phone	✓
Testing notification manager	Allow to notify the mobile phone	✓
Testing GPS tracker	Track the current location of the user	✓
Testing Connection Detector	Detect the network connectivity	✓
Testing GCM Intent Services	Test the Google Cloud Messaging base activities.	✓
Testing Server Utilities	Test Server Base Activities	✓
Testing SQL Database Activities	User details and site details are kept in two databases.	✓
Testing Live Messaging System	GCM base system for send live deals	✓
Testing Google map API base activities	Getting nearby places, getting route for the particular place from Google APIs.	✓
Testing GCM registration	User registration for Google cloud messaging system	✓
Testing JSON Parser	JSON parser to get http requests	✓
Testing Place JSON Parser	JSON parser to get places	✓
Testing Directions JSON Parser	JSON parser to get directions	✓
Testing SQLIte base Database Activities	To keep receiving live message to the mobile	✓

Unit Testing

Robotium is Used for Unit Testing purposes. *Robotium* is an extension of the Android test framework and was created to make it easy to write user interface tests for Android applications

Unit test cases #1 – Testing Live Deals Activity

```
package com.softzone.bingodeals.test;

import com.robotium.solo.Solo;
import com.softzone.bingodeals.MainActivity;
import com.softzone.bingodeals.dbactivity.SiteCategoryActivity;
import com.softzone.bingodeals.livemsg.LiveMsgActivity;
import com.softzone.bingodeals.livemsg.LiveRoutesActivity;
import com.softzone.bingodeals.livemsg.LiveSitesActivity;
```

```
import com.softzone.bingodeals.locnearby.NearSitesActivity;
import com.softzone.bingodeals.locnearby.NearbyPlacesActivity;


import android.test.ActivityInstrumentationTestCase2;

public class InitialButtonTests extends
		ActivityInstrumentationTestCase2<MainActivity> {

	private Solo solo;

	public InitialButtonTests() {
		super("com.softzone.bingodeals",MainActivity.class);
		// TODO Auto-generated constructor stub
	}

	protected void setUp() throws Exception {
		super.setUp();
		solo = new Solo(getInstrumentation(),getActivity());
	}

	//test live Deals
	public void testActivity_1(){
		solo.assertCurrentActivity("Check on main activity ", MainActivity.class);
		solo.clickOnButton("Live Deals!");
		solo.assertCurrentActivity("Check on liveDeals activity ", LiveMsgActivity.class);
		solo.clickInList(0);
		solo.assertCurrentActivity("Check on liveDeals activity ", LiveSitesActivity.class);
		solo.clickOnButton("Get Directions");
		solo.assertCurrentActivity("Check on liveDeals activity ", LiveRoutesActivity.class);
		solo.takeScreenshot("live Routes");
		solo.goBack();
		solo.assertCurrentActivity("Check on liveDeals activity ", LiveSitesActivity.class);
		solo.clickOnButton("Delete Deal");
		solo.assertCurrentActivity("Check on liveDeals activity ", LiveSitesActivity.class);
		solo.goBack();
		solo.assertCurrentActivity("Check on main activity ", MainActivity.class);

	}


}
```

Unit Test Case #2 – Testing Nearby Deals Activity

```
package com.softzone.bingodeals.test;

import com.robotium.solo.Solo;
import com.softzone.bingodeals.MainActivity;
import com.softzone.bingodeals.livemsg.LiveRoutesActivity;
import com.softzone.bingodeals.locnearby.NearSitesActivity;
import com.softzone.bingodeals.locnearby.NearbyPlacesActivity;

import android.test.ActivityInstrumentationTestCase2;

public class DealsAroundButtonTests extends
		ActivityInstrumentationTestCase2<MainActivity> {

	private Solo solo;

	public DealsAroundButtonTests() {
		super("com.softzone.bingodeals",MainActivity.class);
	}

	protected void setUp() throws Exception {
		super.setUp();
		solo = new Solo(getInstrumentation(),getActivity());
	}

	//test live DealsAround
	public void testActivity_1(){
			solo.assertCurrentActivity("Check on main activity ", MainActivity.class);
			solo.clickOnButton("Deals Around");
			solo.assertCurrentActivity("Check on Deals around activity ",
NearbyPlacesActivity.class);
			solo.clickOnButton("Find");
			solo.assertCurrentActivity("Check on NearSitesActivity  ",
NearSitesActivity.class);
			solo.clickOnButton("Get Directions");
			solo.assertCurrentActivity("Check on LiveRoutesActivity activity ",
LiveRoutesActivity.class);
			solo.goBack();
			solo.assertCurrentActivity("Check on NearSitesActivity activity ",
NearSitesActivity.class);
			solo.goBack();
			solo.assertCurrentActivity("Check on NearbyPlacesActivity activity ",
NearbyPlacesActivity.class);
			solo.goBack();
			solo.assertCurrentActivity("Check on main activity ", MainActivity.class);
```

Conclusion

Several Performance aspects were considered when creating the Bingo Deals Application

Extensibility

Reliability

Portability

Security

Response time in website

- Response time in website is less than 1 second. User can to register, login and select types from site as quickly as possible

Response time in Bingo application

Response time in Bingo application is less than 5 seconds. Deals sent to the application within 5 seconds

Concurrent Access

At least 100 users can be able to concurrently access to the website. Deals can be sent to maximum 100 people at once.

Response time to access database activites

Maximum Response time to access database activites is 3 seconds in 2G network. It is very less when using 3G netwok.

References

[1]. https://developers.google.com/maps/documentation/android/intro

[2]. https://developers.google.com/places/documentation/index

[3]. http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java

[4]. http://developer.android.com/google/gcm/gcm.html

[5]. http://www.androidhive.info/

[6]. https://developer.android.com/google/gcm/index.html

[7]. http://stackoverflow.com/

[8]. http://en.wikipedia.org/wiki/Global_Positioning_System

[9]. http://developer.android.com/about/index.html

[10]. http://en.wikipedia.org/wiki/Eclipse_%28software%29