Group 10 Design Specification

February 15, 2014

SE.10.D3

Version: 1.6
Status: Release

Contributor Name	Role
Daniel Clark	Project Lead
Mark Lewis	QA Manager
Charles Newey	Deputy Project Lead & Android Developer
Martin Ferris	Android Developer
Ashley Iles	Android Developer
Kenny Packer	Android Developer
Stephen McFarlane	Deputy QA & Web Developer
Kieran Palmer	Web Developer

Department of Computer Science,
Llandinam Building,
Aberystwyth University,
Aberystwyth,
Ceredigion,
SY23 3DB

Contents

1 INTRODUCTION **4**
1.1 Purpose of This Document . . . 4
1.2 Scope . . . 4
1.3 Objectives . . . 4

2 DECOMPOSITION DESCRIPTION **4**
2.1 Programs in System . . . 4
2.2 Significant Classes - Android . . . 5
2.2.1 StumblrData . . . 5
2.2.2 Waypoint . . . 5
2.2.3 Route . . . 5
2.3 Significant Activities - Android . . . 5
2.3.1 AbstractActivity . . . 5
2.3.2 CreateWaypoint . . . 5
2.3.3 CreateRoute . . . 5
2.3.4 GPSService . . . 5
2.3.5 WaypointList . . . 5
2.3.6 FinishRoute . . . 5
2.3.7 Home . . . 6
2.4 Significant Classes - Web . . . 6
2.4.1 Page . . . 6
2.4.2 Database Interaction . . . 6
2.5 Shared Modules Between Programs . . . 6
2.6 Mapping Requirements to Classes . . . 7

3 DEPENDENCY DESCRIPTION **8**
3.1 Class Diagrams - Inheritance Relationships . . . 8
3.1.1 Android Activity Classes . . . 8
3.1.2 Android Data Classes . . . 9

4 INTERFACE DESCRIPTION **10**
4.1 Android Activities . . . 10
4.1.1 AbstractActivity {abstract} . . . 10
4.1.2 Home . . . 12
4.1.3 CreateRoute . . . 13
4.1.4 CreateWaypoint . . . 15
4.1.5 GPSService . . . 18
4.1.6 WaypointList . . . 21
4.1.7 FinishRoute . . . 25
4.2 Data Classes . . . 27
4.2.1 StumblrData {abstract} . . . 27
4.2.2 Route . . . 29
4.2.3 Waypoint . . . 33

5 DETAILED DESIGN **36**
5.1 Sequence Diagram . . . 36
5.2 Significant Algorithms . . . 37
5.2.1 Upload Data . . . 37
5.2.2 Bundle Data . . . 37
5.2.3 Validate Inputs . . . 37
5.2.4 Change Screen . . . 37
5.3 Significant Data Structures - Android . . . 37
5.3.1 Route . . . 37

5.3.2 Waypoint . . . 37
5.3.3 Co-ordinates . . . 37
5.4 Significant Data Structures - MySQL Database . . . 38
5.4.1 List of Walks . . . 38
5.4.2 Location . . . 38
5.4.3 Place Description . . . 38
5.4.4 Photo Usage . . . 38
5.5 Website Interface Specification . . . 38
5.5.1 Page . . . 38
5.5.2 Map . . . 39
5.5.3 Database Interaction . . . 39
5.6 JSON Example . . . 40
5.7 HTTP Transaction and Expected Values . . . 40

6 Android Design Justifications **41**

7 Web Design Justifications **41**

8 REFERENCES **42**

9 VERSION HISTORY **42**

1 INTRODUCTION

1.1 Purpose of This Document

The purpose of this document is to ensure that the developers have a complete and holistic understanding of how the project is going to be structured and evolve into a fully functional system. This will be done by establishing the key aspects and identifying each unique requirement stated initinally by the client. Following are the key aspects such as descriptions of classes and how the interface is going to be presented to the user and to specify the procedures taken to ensure their completion.

1.2 Scope

This design specification will separate the project into components which are implemented separately from each other - describing the high-level interactions between components.

1.3 Objectives

- To provide a description of the main components of Stumblr.
- To provide an explanation of the dependencies between certain components, describing for both compilation and execution.
- To provide details about the interface for all main classes of Stumblr.

2 DECOMPOSITION DESCRIPTION

2.1 Programs in System

The System that will be created will contain three systems; the Android app, a server and a web page.

The Android side of the system will be used as the platform for the walking tour creator app. Within this app users can create a new tour, they will then be able to add locations to the tour. These locations will contain information such as the tours title, description, the way points of the location and possibly an image. Once the tour is saved and finished it can then be sent via Wi-Fi or internet connection on the device to the server. The data will be sent to the server as a Multimedia Internet Message Extension (MIME) format in an HTTP POST request.

The server will be used to store all data about tours created on the Android app. On the server a database will be created and within the database a table for storing the tour's details. Once data is received a new record will be created and the data stored for easy re-use on the web page. To access this data the web page will use a PHP script with MySQL commands.

The final program in the system is the web page, which will be used for displaying tours. On the website there will be a list of saved tours from which the user can click to view them. The tour will then be displayed on a map which will incorporate the use of the Google Maps API. By clicking on a Waypoint the user will be able to view the information added by the creator.

2.2 Significant Classes - Android

2.2.1 StumblrData

This will be an abstract class which will contain variables and validation methods that other classes will use, such as "short description" and "title".

2.2.2 Waypoint

This class will handle all information and methods regarding each individual Waypoint on the route, including acquisition and processing of image and GPS data from the Android system. A constructor method will be used to create a Waypoint containing "title", "short description" and an image. Coordinates will be stored in a LinkedList of (android.location.Location) objects. The LinkedList will contain the list of coordinates between the current and previous Waypoints.

2.2.3 Route

This class will contain all the methods to handle all parts of creating a route, such as adding Waypoints to a linked list of Waypoints.

2.3 Significant Activities - Android

2.3.1 AbstractActivity

AbstractActivity will hold all of the common methods (common bundle loading/saving techniques). Each other activity will inherit from it.

2.3.2 CreateWaypoint

Inherits from DataEntryActivity. Contains methods to set images, timestamps, GPS coordinates, and other information in the constructor.

2.3.3 CreateRoute

Inherits from DataEntryActivity. Contains methods to create a new Route LinkedList and access items within the LinkedList. Also contains methods to access data held within text fields - from the corresponding data entry field.

2.3.4 GPSService

A simple foreground service that serves as a way of obtaining GPS cooordinates without being killed by the Android system.

2.3.5 WaypointList

Inherits from DataEntryActivity. This activity contains methods and UI elements to show the Waypoint objects in a list-type structure within the app.

2.3.6 FinishRoute

Inherits from DataEntryActivity. This contains methods to package the Route object into something that can be uploaded (via JSON) to the server application, as well as giving the user a persistent notification icon showing the status of their upload.

2.3.7 Home

Inherits from DataEntryActivity. This is the main screen of the application, that shows all of the menu options necessary (and also the Settings menu). This will contain methods to progress to the CreateRoute activity.

2.4 Significant Classes - Web

2.4.1 Page

The main web interface will make use of the Map and the Database Interaction files. The main interface will employ JavaScript, HTML, and CSS. This page will show the map, using the database interaction file to pull the relevant information from the MySQL database, and render it on-screen using the Google Maps API and JavaScript.

2.4.2 Database Interaction

The database interaction file will abstract the database interaction away from the main web interface page; it will be used to add and request data from the MySQL database. Included in the database will be the locations for the Maps API, as well as showing different tours in a table next to it markers for each route's Waypoints in another table.

2.5 Shared Modules Between Programs

As the two programs differ enormously in terms of the technology used, there are no common modules, per se. However, one could consider the fact that both programs are expected to communicate using the HTTP protocol.

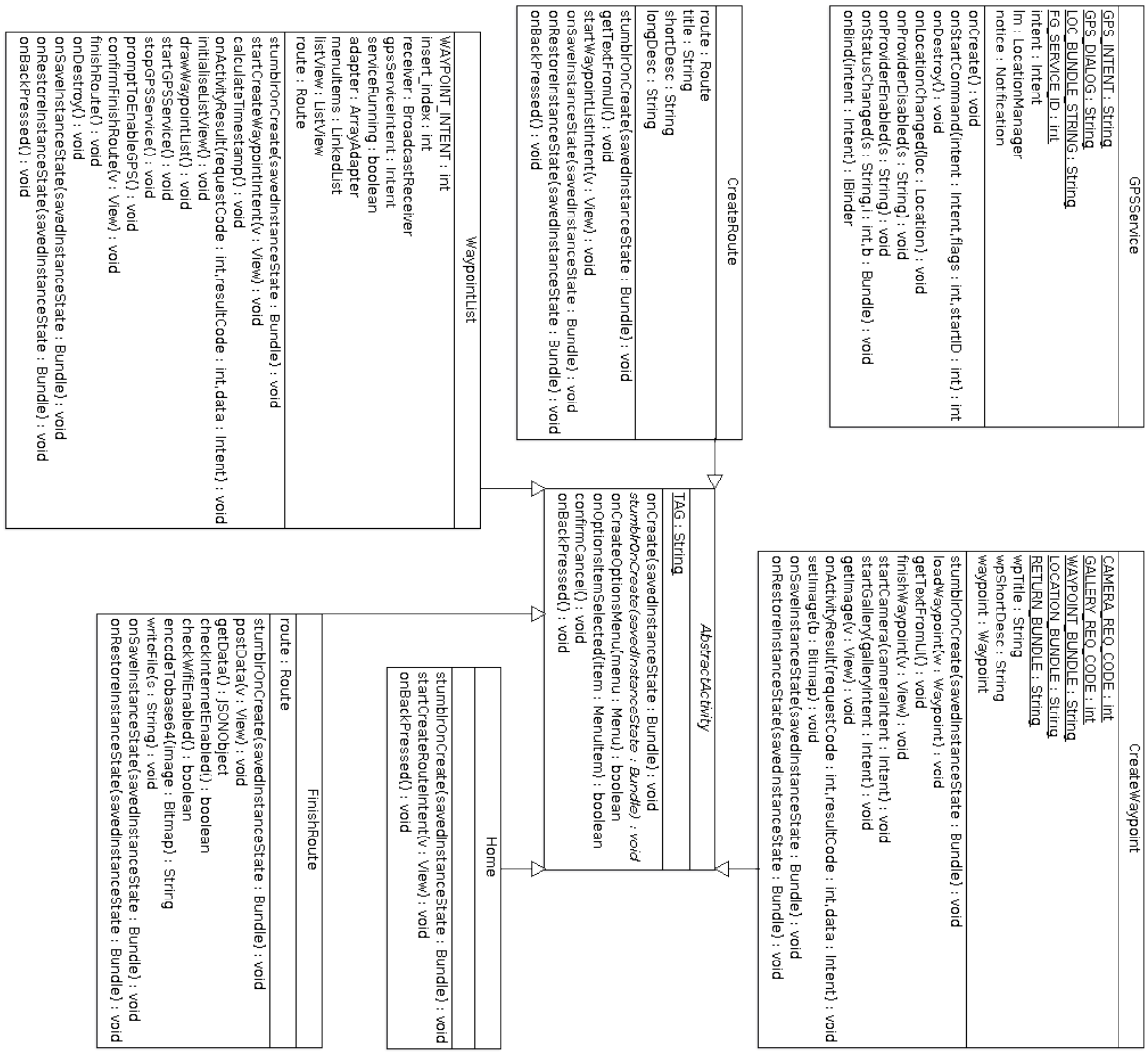
2.6 Mapping Requirements to Classes

1. **FR1; Startup of software on Android device**
   Android class Home and Android class AbstractActivity.
2. **FR2; Providing info about the whole walking tour**
   Android class WaypointList and AbstractActivity.
3. **FR3; Adding locations to the walking tour**
   Android class CreateWaypoint, and Android class AbstractActivity.
4. **FR4; Adding photos to the walks**
   Android class CreateWaypoint and Android class AbstractActivity.
5. **FR5; Cancelling walks**
   Android class CreateWaypoint and Android class AbstractActivity.
6. **FR6; Sending the walk to the server**
   Android class FinishRoute and Android class AbstractActivity.
7. **FR7; Switching from the WTC**
   Android class AbstractActivity holds the basic code for switching between apps. Each individual Activity class contains its own code in **onSavedInstanceState()** methods.
8. **FR8; Trying out a created walking tour**
   Web Page, Map and Database Interaction.
9. **FR9; Saving data on server**
   Android class FinishRoute, Android class AbstractActivity, Web Page, Map and Database Interaction.

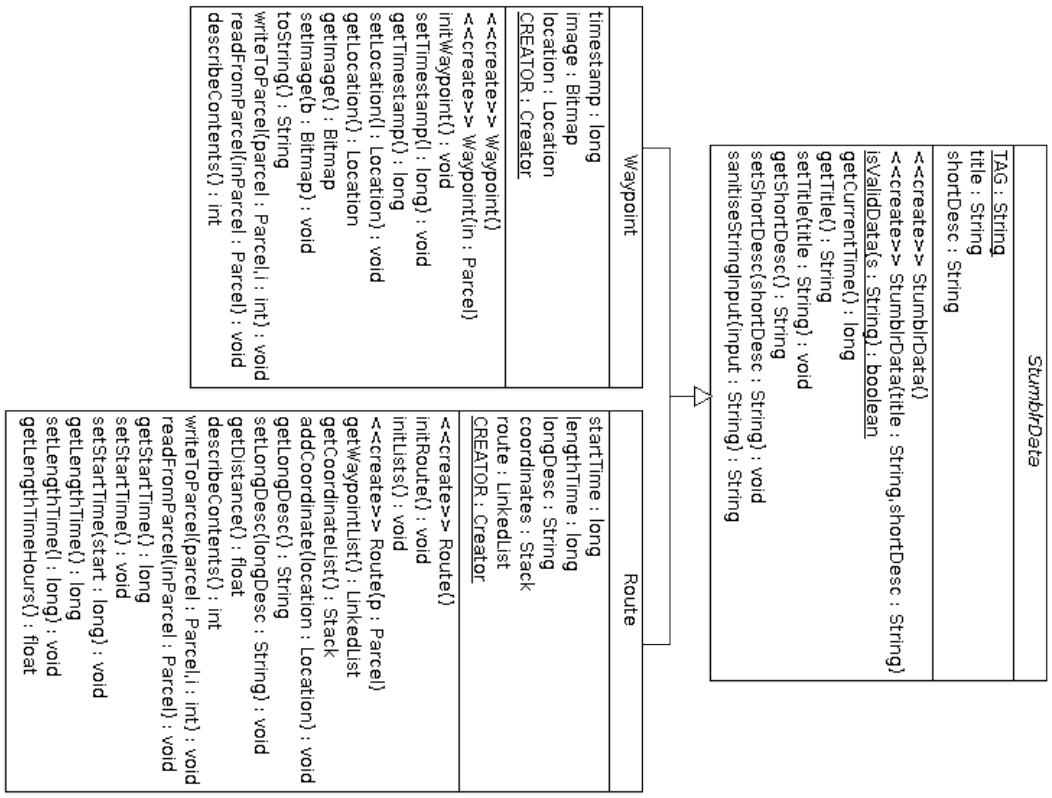
3 DEPENDENCY DESCRIPTION

3.1 Class Diagrams - Inheritance Relationships

3.1.1 Android Activity Classes



3.1.2 Android Data Classes



4 INTERFACE DESCRIPTION

4.1 Android Activities

4.1.1 AbstractActivity {abstract}

Listing 1: AbstractActivity {Abstract}

```
1  package uk.ac.aber.cs.groupten.stumblr;
2
3  import android.app.AlertDialog;
4  import android.content.DialogInterface;
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.support.v7.app.ActionBarActivity;
8  import android.view.Menu;
9  import android.view.MenuItem;
10
11 public abstract class AbstractActivity extends ActionBarActivity {
12     /**
13      * Logging tag.
14      */
15     public static final String TAG = "STUMBLR";
16
17     /**
18      * @param savedInstanceState saved state of the application.
19      * onCreate set state to saved previously frozen state.
20      */
21     @Override
22     protected void onCreate(Bundle savedInstanceState);
23
24     /**
25      * @param savedInstanceState
26      * Called by super class
27      */
28     public abstract void stumblrOnCreate(Bundle savedInstanceState);
29
30     /**
31      * @param menu inflate the menu; this adds items to the action bar if
32          it is present.
33      * @return true once menu has inflated.
34      */
35     @Override
36     public boolean onCreateOptionsMenu(Menu menu);
37
38     /**
39      * Handle action bar item clicks here. The action bar will
40          automatically handle
41      * clicks on the Home/Up button, so long as you specify a parent
42          activity in
43      * AndroidManifest.xml
44      * @param item passed in MenuItem when it is selected
45      * @return selected item
46      */
47     @Override
```

```
45      public boolean onOptionsItemSelected(MenuItem item);

47      public void confirmCancel();

49      /**
         * When back button is pressed, call finish. Drop activity from memory
            .
51       */
        @Override
53      public void onBackPressed();
}
```

4.1.2 Home

Listing 2: Home

```
package uk.ac.aber.cs.groupten.stumblr;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class Home extends AbstractActivity {
    /**
     * Loads the activity on creation (using a bundle if one is present)
     * @param savedInstanceState The bundle containing the saved instance
        state.
     */
    public void stumblrOnCreate(Bundle savedInstanceState);

    /**
     * Begin the Route entry activity
     */
    public void startCreateRouteIntent(View v);

    /**
     * When back is pressed, finish activity.
     */
    @Override
    public void onBackPressed();
}
```

4.1.3 CreateRoute

Listing 3: CreateRoute

```
package uk.ac.aber.cs.groupten.stumblr;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import uk.ac.aber.cs.groupten.stumblr.data.Route;
import uk.ac.aber.cs.groupten.stumblr.data.StumblrData;

public class CreateRoute extends AbstractActivity {

    /**
     * Instance of Route.
     */
    private Route route;

    /**
     * Title of route.
     */
    private String title;

    /**
     * Short description of route.
     */
    private String shortDesc;

    /**
     * Long description of route.
     */
    private String longDesc;

    /**
     * Loads the activity on creation (using a bundle if one is present)
     *
     * @param savedInstanceState The bundle containing the saved instance
        state.
     */
    @Override
    public void stumblrOnCreate(Bundle savedInstanceState);

    /**
     * Get text from fields in the user interface and sanitise inputs.
     */
    public void getTextFromUI();

    /**
     * Start the WaypointList activity (list the current Route).
     * Called when the "next" button is clicked in the UI.
```

```
     */
52   public void startWaypointListIntent(View v);

54   /**
     * On instance saved, call getTextFromUI() and set the strings to
        appropriate variables.
56   * @param savedInstanceState
     */
58   @Override
     public void onSaveInstanceState(Bundle savedInstanceState);
60
     /**
62   * Restore instance state to savedInstanceState. Set title, shortDesc
        and longDesc by getting
     * them from the savedInstanceState. Then set textview fields to their
         appropriate variables.
64   * @param savedInstanceState
     */
66   @Override
     public void onRestoreInstanceState(Bundle savedInstanceState);
68
     /**
70   * Logs when back is pressed in CreateRoute.
     */
72   @Override
     public void onBackPressed()
74 }
```

4.1.4 CreateWaypoint

Listing 4: CreateWaypoint

```
package uk.ac.aber.cs.groupten.stumblr;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.location.Location;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.view.WindowManager;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import java.io.FileNotFoundException;
import java.io.IOError;
import java.io.IOException;
import java.util.Calendar;

import uk.ac.aber.cs.groupten.stumblr.data.StumblrData;
import uk.ac.aber.cs.groupten.stumblr.data.Waypoint;

public class CreateWaypoint extends AbstractActivity {
    /**
     * Request code for camera intent
     */
    public static final int CAMERA_REQ_CODE = 1337;

    /**
     * Request code for gallery intent
     */
    public static final int GALLERY_REQ_CODE = 7007;

    /**
     * Bundle string for identifying Waypoint data
     */
    public static final String WAYPOINT_BUNDLE = "waypoint";

    /**
     * Bundle string for identifying location data
     */
    public static final String LOCATION_BUNDLE = "loc";

    /**
     * Bundle string for identifying return data
     */
    public static final String RETURN_BUNDLE = "return_data";
```

```
52
54      /**
        * Waypoint title.
        */
56      private String wpTitle;

58      /**
        * Short description.
60      */
        private String wpShortDesc;
62
        /**
64      * Instance of waypoint.
        */
66      private Waypoint waypoint;

68      /**
        * Loads the activity on creation (using a bundle if one is present)
70      *
        * @param savedInstanceState The bundle containing the saved instance
            state.
72      */
        public void stumblrOnCreate(Bundle savedInstanceState);
74
        /**
76      * Pass in waypoint. Set TextView and ImageView fields to data
            returned from getting
        * the waypoints title, short description and image
78      * @param w passed in waypoint to be loaded.
        */
80      public void loadWaypoint(Waypoint w);

82      /**
        * Gets waypoint title and short description from the user interface.
84      */
        public void getTextFromUI();
86
        /**
88      * Called when "Create" button in the UI is clicked.
        * Adds data to the current Waypoint object with text specified in UI.
90      */
        public void finishWaypoint(View v);
92
        /*
94      * ****************************************************************
        *                          Camera interaction                     *
96      * ****************************************************************
        */
98      /**
        * Obtain a photo from user and add it to current Waypoint.
100     */
        public void startCamera(Intent cameraIntent);
102
        public void startGallery(Intent galleryIntent);
```

```
104
        public void getImage(View v);
106
        /**
108      * on Creation of a waypoint result, if the result is OK then
             getExtras from intent data.
         * Then get bitmap image for waypoint image and check for null prior
             to setting the image to the
110      * waypoint. Update UI imageView, and log dimensions of image.
         *
112      * @param requestCode
         * @param resultCode
114      * @param data
         */
116     @Override
        protected void onActivityResult(int requestCode, int resultCode,
            Intent data);
118
        public void setImage(Bitmap b);
120
        @Override
122     public void onSaveInstanceState(Bundle savedInstanceState);

124     /**
         * Restore state to savedInstanceState. Set waypoint title, short
             description and waypoint
126      * to appropriate data retrieved from the savedInstanceState. Then
             update TextViews with
         * newly set title and short description.
128      * @param savedInstanceState
         */
130     @Override
        public void onRestoreInstanceState(Bundle savedInstanceState);
132 }
```

4.1.5 GPSService

Listing 5: DataEntryActivity {Abstract}

```
package uk.ac.aber.cs.groupten.stumblr;

import android.app.Notification;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Build;
import android.os.Bundle;
import android.os.IBinder;
import android.support.v4.app.NotificationCompat;

public class GPSService extends Service implements LocationListener {

    /**
     * String name for GPS_INTENT.
     */
    public final static String GPS_INTENT = "STUMBLR_GPS";

    /**
     * String name for GPS_DIALOG.
     */
    public final static String GPS_DIALOG = "STUMBLR_GPS_DIALOG";

    /**
     * String name for bundle of location strings.
     */
    public final static String LOC_BUNDLE_STRING = "loc";

    /**
     * Service ID.
     */
    private static final int FG_SERVICE_ID = 101;

    /**
     * Instance of Intent.
     */
    private Intent intent;

    /**
     * Instance of LocationManager.
     */
    private LocationManager lm;

    /**
     * Instance of Notification.
     */
    private Notification notice;
```

```
52
54      /**
        * onCreate start new intent for GPS_INTENT
        */
56      @Override
        public void onCreate();
58
        /**
60      * @param intent  The intent that the Service was started from
        * @param flags   Startup flags
62      * @param startID Service ID
        * @return The service status (Sticky, non-sticky, etc)
64      */
        @Override
66      public int onStartCommand(Intent intent, int flags, int startID);

68      /**
        * onDestroy remove LocationManager updates and stop foreground task.
70      */
        @Override
72      public void onDestroy();

74      /**
        * Obtain coordinates from Android system and add to current Waypoint.
76      */
        @Override
78      public void onLocationChanged(Location loc);

80      /**
        * Prompt for GPS, and error handling
82      */
        @Override
84      public void onProviderDisabled(String s);

86      /**
        * Empty method to abide by implementation requirements.
88      * @param s
        */
90      @Override
        public void onProviderEnabled(String s);
92
        /**
94      * Empty method to abide by implementation requirements.
        * @param s
96      * @param i
        * @param b
98      */
        @Override
100     public void onStatusChanged(String s, int i, Bundle b);

102     /**
        * Empty method to abide by implementation requirements.
104     * @param intent
        * @return
```

```
106         */
        public IBinder onBind(Intent intent); // Unused
108 }
```

4.1.6 WaypointList

Listing 6: WaypointList

```
package uk.ac.aber.cs.groupten.stumblr;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.location.Location;
import android.os.Bundle;
import android.provider.Settings;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.EmptyStackException;
import java.util.LinkedList;

import uk.ac.aber.cs.groupten.stumblr.data.Route;
import uk.ac.aber.cs.groupten.stumblr.data.Waypoint;

public class WaypointList extends AbstractActivity {

    /**
     * Waypoint intent constant.
     */
    private final int WAYPOINT_INTENT = 3141;

    /**
     * Index to insert new waypoint.
     */
    private int insert_index;

    /**
     * GPS Broadcast reciever.
     */
    private BroadcastReceiver receiver;

    /**
     * GPS Service intent.
     */
    private Intent gpsServiceIntent;

    /**
     * Boolean to show if GPS service is running.
     */
    private boolean serviceRunning = false;
```

```
52      // ListView objects
        /**
54       * Adapter used to put menu items into linked list
         */
56      private ArrayAdapter<Waypoint> adapter;

58      /**
         * List of menu items.
60       */
        private LinkedList<Waypoint> menuItems;
62
        /**
64       * View of the list.
         */
66      private ListView listView;

68      /**
         * Instance of route.
70       */
        private Route route;
72
        /**
74       * Loads the activity on creation (using a bundle if one is present)
         *
76       * @param savedInstanceState The bundle containing the saved instance
             state.
         */
78      public void stumblrOnCreate(Bundle savedInstanceState);

80      /**
         * Starts the CreateWaypoint Intent, so that it returns a result.
82       *
         * @param v The View object.
84       */
        public void startCreateWaypointIntent(View v);
86
        /**
88       * Calculate time by subtracting the endTime by the startTime.
         * Set the length route was recorded for to the route.
90       */
        private void calculateTimestamp();
92
        /**
94       * Called when an Activity is dispatched for a result
         *
96       * @param requestCode Integer relating the intent to a request
         * @param resultCode  Used to check if a request was successful
98       * @param data        The Intent used
         */
100     @Override
        protected void onActivityResult(int requestCode, int resultCode,
            Intent data);
102
        /**
```

```
104         * Initialises the ListView and Adapter objects.
            */
106         private void initialiseListView();

108         /**
            * Renders Waypoint list on screen.
110         */
            public void drawWaypointList();
112
            /**
114         * Starts the GPS service.
            */
116         private void startGPSService();

118         /**
            * Halts the GPS service.
120         */
            private void stopGPSService();
122
            /**
124         * Display dialog to prompt the user to enable GPS. Then take them to
              the settings page.
            */
126         public void promptToEnableGPS();

128         /**
            * Finish route method. Display message to check if the user is
              finished. If button is pressed
130         * call finishRoute and then finish the activity.
            */
132         public void confirmFinishRoute(View v);

134         /**
            * Passes the current Route object to FinishRoute and starts the
              activity.
136         * @param v The View object passed in by the Android OS.
            */
138         public void finishRoute() {

140         /**
            * onDestroy, stop the GPS service.
142         */
            @Override
144         public void onDestroy();
            /**
146         * When instance state is saved, put instance variables into it as
              well as route data.
            * @param savedInstanceState
148         */
            @Override
150         public void onSaveInstanceState(Bundle savedInstanceState);

152         /**
            * When instance state is restored, retrieve instance variables and
```

```
154         * data from saved instance state.
            * @param savedInstanceState
156         */
        @Override
158     public void onRestoreInstanceState(Bundle savedInstanceState);

160     /**
         * Log when back is pressed.
162      */
        @Override
164     public void onBackPressed();
    }
```

4.1.7 FinishRoute

Listing 7: FinishRoute

```
1  package uk.ac.aber.cs.groupten.stumblr;
2
3  import android.app.AlertDialog;
4  import android.app.Dialog;
5  import android.content.Context;
6  import android.content.DialogInterface;
7  import android.content.Intent;
8  import android.graphics.Bitmap;
9  import android.location.Location;
10 import android.net.ConnectivityManager;
11 import android.net.NetworkInfo;
12 import android.os.AsyncTask;
13 import android.os.Bundle;
14 import android.provider.Settings;
15 import android.util.Base64;
16 import android.util.Log;
17 import android.view.View;
18 import android.widget.TextView;
19 import android.widget.Toast;
20
21 import org.apache.http.HttpResponse;
22 import org.apache.http.client.HttpClient;
23 import org.apache.http.client.methods.HttpPost;
24 import org.apache.http.entity.StringEntity;
25 import org.apache.http.impl.client.DefaultHttpClient;
26 import org.json.JSONArray;
27 import org.json.JSONException;
28 import org.json.JSONObject;
29
30 import java.io.ByteArrayOutputStream;
31 import java.io.FileNotFoundException;
32 import java.io.FileOutputStream;
33 import java.io.IOException;
34 import java.io.OutputStreamWriter;
35 import java.lang.reflect.Method;
36 import java.util.LinkedList;
37 import java.util.Stack;
38
39 import uk.ac.aber.cs.groupten.stumblr.data.Route;
40 import uk.ac.aber.cs.groupten.stumblr.data.Waypoint;
41
42 public class FinishRoute extends AbstractActivity {
43     private Route route;
44
45     /**
46      * Loads the activity on creation (using a bundle if one is present)
47      *
48      * @param savedInstanceState The bundle containing the saved instance
49          state.
50      */
51     public void stumblrOnCreate(Bundle savedInstanceState);
```

```
51
        /*
53       * ****************************************************************
         *                        HTTP POST Interaction                   *
55       * ****************************************************************
         */
57      private class NetworkTask extends AsyncTask<String, Void, HttpResponse
            >;
             protected void onPostExecute(HttpResponse result); // Private
                method
59
        /**
61       * Posts the data to the server. Check if internet is available first.
         */
63      public void postData(View v);

65      private JSONObject getData();
        // End HTTP POST
67
        /**
69       * Ensuring Network Provider is Enabled before submitting route
         */
71      public boolean checkInternetEnabled();

73      public boolean checkWifiEnabled();

75      /*
         * ****************************************************************
77       *                          Base64 Encoding                       *
         * ****************************************************************
79       */
        public String encodeTobase64(Bitmap image);
81
        // File writing
83      public void writeFile(String s);

85      @Override
        public void onSaveInstanceState(Bundle savedInstanceState);
87
        @Override
89      public void onRestoreInstanceState(Bundle savedInstanceState);
    }
```

4.2 Data Classes

4.2.1 StumblrData {abstract}

Listing 8: StumblrData

```
package uk.ac.aber.cs.groupten.stumblr.data;

import android.os.Parcelable;

import java.util.Calendar;

/**
 * Abstract class containing basic common structure for all Stumblr data formats.
 */
public abstract class StumblrData implements Parcelable {
    public static final String TAG = "STUMBLR";

    /**
     * The title of the given piece of data. This can be extended to Waypoints, Routes
     * and any other piece of relevant data inside the structure of Stumblr.
     */
    private String title;

    /**
     * The short description pertaining to the given piece of data. This will also be
     * extended to other component classes of StumblrData
     */
    private String shortDesc;

    /**
     * Default constructor
     */
    public StumblrData() {
        // do nothing
    }

    /**
     * @param title     The title to set.
     * @param shortDesc The short description to set.
     */
    public StumblrData(String title, String shortDesc);

    /**
     * Checks the StumblrData item for validity. Returns a boolean. (true = valid)
     *
     * @return Whether the data is valid or not. (true = valid)
     * <p/>
     * MUST be implemented in any subclasses.
     */
```

```
    public static boolean isValidData(String s);

    /**
     * Returns current time.
     *
     * @return The current time.
     */
    public long getCurrentTime();

    /**
     * Returns the title.
     *
     * @return this.title
     */
    public String getTitle();

    /**
     * Sets the current title.
     *
     * @param title The title to set.
     */
    public void setTitle(String title);

    /**
     * Returns the short description.
     *
     * @return shortDesc
     */
    public String getShortDesc();

    /**
     * Sets the short description.
     *
     * @param shortDesc
     */
    public void setShortDesc(String shortDesc);

    /**
     * Sanitises given text by removing prohibited characters.
     *
     * @param input The text to sanitise.
     * @return The sanitised string.
     */
    public String sanitiseStringInput(String input);
}
```

4.2.2 Route

Listing 9: Route

```
1  package uk.ac.aber.cs.groupten.stumblr.data;

3  import android.location.Location;
   import android.os.Parcel;
5  import android.os.Parcelable;
   import android.util.Log;
7
   import java.util.LinkedList;
9  import java.util.Stack;

11 public class Route extends StumblrData implements Parcelable {

13     /**
        * startTime used for timestamp of start of the walk.
15      */
       private long startTime;
17
       /**
19      * lengthTime used for timestamp for length of walk.
        */
21     private long lengthTime;

23     /**
        * longDesc
25      * A slightly longer description of the contents of the route. Set by
           the user when
        * they create a Route.
27      */
       private String longDesc;
29
       /**
31      * The list of Location objects that reflect the coordinates of the
           walk.
        */
33     private Stack<Location> coordinates;

35     /**
        * A LinkedList of Waypoint objects that the Route comprises of.
37      */
       private LinkedList<Waypoint> route;
39
       /**
41      * Constructor. Calls initRoute() to initialise route.
        */
43     public Route();

45     /**
        * Helper method for constructor. Calls initLists() to initialise the
           list and stack.
47      */
       private void initRoute();
```

```
49
50      /**
51       * List and stack initializer method.
52       */
53      private void initLists();
54
55      /**
56       * Initialise Route object from a Parcel.
57       */
58      public Route(Parcel p);
59
60      /**
61       * @return The LinkedList of Waypoint objects.
62       */
63      public LinkedList<Waypoint> getWaypointList();
64
65      /**
66       * @return The list of coordinates.
67       */
68      public Stack<Location> getCoordinateList();
69
70      /**
71       * Adds an item to the list of coordinates.
72       * @param location The Location object to add.
73       */
74      public void addCoordinate(Location location);
75
76      /**
77       * Returns the long description of the Route.
78       * @return The long description of the Route.
79       */
80      public String getLongDesc();
81      /**
82       * Sets the long description.
83       *
84       * @param longDesc A longer description of the Route.
85       */
86      public void setLongDesc(String longDesc) {
87          this.longDesc = longDesc;
88      }
89
90
91      /**
92       * Method loops through coordinate list.
93       *
94       * If list is empty, return 0.0f. Sets current location by calling
             getLatitude()
95       * and getLongitude() methods. On each loop, increment i and set
             currentLoc to coordinates
96       * held in position i of coordinates list. Do same for nextLoc but add
               1 so it retrieves the
97       * coordinates of the position in front of current location. Using
             distanceBetween() it
98       * calculates distance between the two locations and add results to
```

```
         distance variable.
99       *
         * @return returns the route distance.
101      */
       public float getDistance();
103
       /**
105     * Describes contents of parcel.
         * @return Description
107      */
       @Override
109    public int describeContents();

111    /**
         * Writes the Route into a Parcel for moving between screens.
113      *
         * @param parcel The parcel to be written to.
115      * @param i      Flags.
         */
117    @Override
       public void writeToParcel(Parcel parcel, int i);
119
       /**
121     * Reads Route data from a parcel.
         *
123      * @param inParcel The parcel to read the Route from.
         */
125    public void readFromParcel(Parcel inParcel);

127    /**
         * Private class that helps create a Parcelable.
129      */
       public static final Parcelable.Creator<Route> CREATOR = new Parcelable
          .Creator<Route>();
131
       /**
133     * @return startTime of the walk
         */
135    public long getStartTime();

137    /**
         * startTime gets set to current time.
139      */
       public void setStartTime();
141
       /**
143     * @param start pass in start time
         */
145    public void setStartTime(long start);

147    /**
         * @return lengthTime variable
149      */
       public long getLengthTime();
```

```
151
        /**
153      * @param l pass in length time and set lengthTime to it.
         */
155     public void setLengthTime(long l);

157     /**
         * @return lengthTime as a casted float divided to be hours.
159      */
        public float getLengthTimeHours();
161 }
```

4.2.3 Waypoint

Listing 10: Waypoint

```
1  package uk.ac.aber.cs.groupten.stumblr.data;

3  import android.graphics.Bitmap;
   import android.location.Location;
5  import android.os.Parcel;
   import android.os.Parcelable;
7  import android.util.Log;

9  public class Waypoint extends StumblrData implements Parcelable {
       // CONSTRUCTORS
11
       /**
13      * Default constructor for Waypoint.
        */
15     public Waypoint();

17     /**
        * Constructor for a Waypoint object from a Parcel.
19      */
       public Waypoint(Parcel in);
21
       // INSTANCE VARIABLES
23
       /* title and shortDesc are declared in StumblrData and are accessed
          through get/set methods */
25     /**
        * Arrival timestamp for Waypoint.
27      */
       private long timestamp;
29
       /**
31      * Image contained within Waypoint.
        */
33     private Bitmap image;

35     /**
        * Location for Waypoint
37      */
       private Location location;
39
       /**
41      * Helper method for initialising Waypoint objects.
        */
43     private void initWaypoint();

45     /**
        * Sets timestamp.
47      *
        * @param l The timestamp.
49      */
       public void setTimestamp(long l);
```

```
51
52      /**
53       * Returns timestamp.
54       *
55       * @return The timestamp.
56       */
57      public long getTimestamp();
58
59      /**
60       * Sets the current location.
61       *
62       * @param l The current location to set.
63       */
64      public void setLocation(Location l);
65
66
67      /**
68       * Sets the current location.
69       *
70       * @param l The current location to set.
71       */
72      public Location getLocation();
73
74      /**
75       * Returns current Bitmap.
76       *
77       * @return The current Bitmap that the Waypoint has,
78       */
79      public Bitmap getImage();
80
81      /**
82       * Sets the current Bitmap.
83       *
84       * @param b The current Bitmap.
85       */
86      public void setImage(Bitmap b);
87
88      /**
89       * Returns a String with the title.
90       *
91       * @return The title string.
92       */
93      public String toString();
94
95      /**
96       * Writes the Waypoint into a Parcel for moving between Activities.
97       *
98       * @param parcel The parcel to be written to.
99       * @param i      Flags.
100      */
101     @Override
102     public void writeToParcel(Parcel parcel, int i);
103
104     /**
```

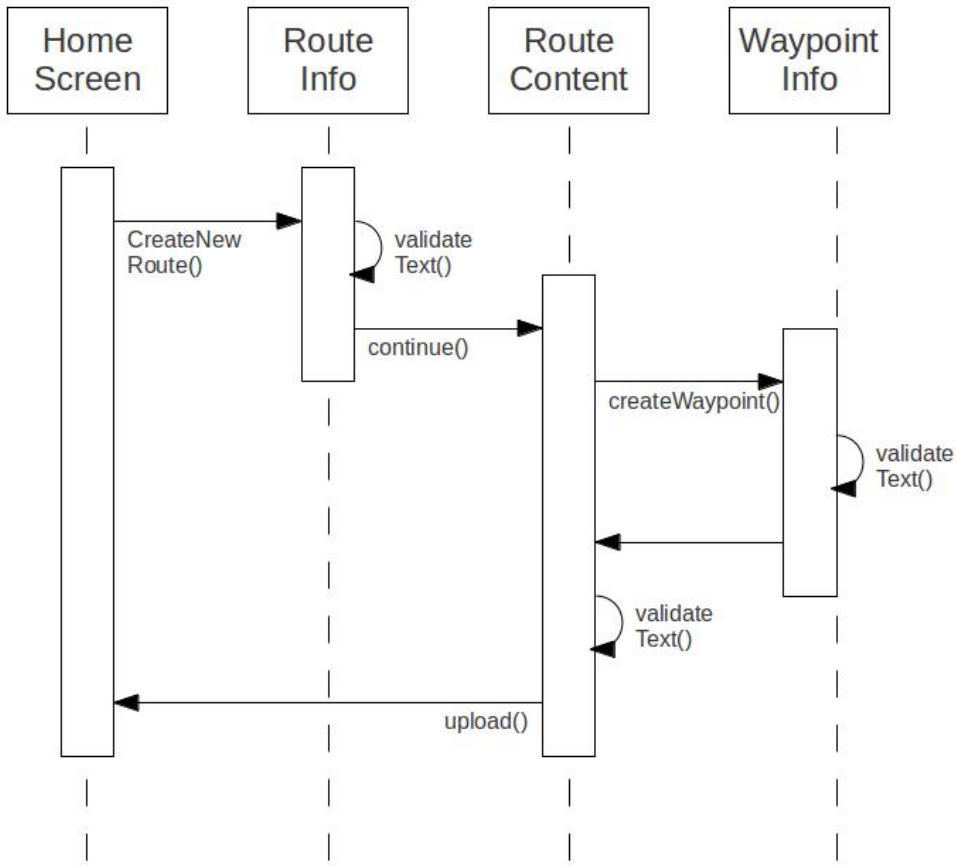
```
105         * Reads Route data from a parcel.
            *
107         * @param inParcel
            */
109        public void readFromParcel(Parcel inParcel);

111        public static final Parcelable.Creator<Waypoint> CREATOR = new
               Parcelable.Creator<Waypoint>();
            public Waypoint createFromParcel(Parcel in);
113
            /**
115          * @param size pass in size of new array.
             * @return Waypoint type array with passed in size.
117          */
            public Waypoint[] newArray(int size);
119        };

121       @Override // Unused
          /**
123        * Unused method. Required for parcelable implementation.
           */
125       public int describeContents();
    }
```

5 DETAILED DESIGN

5.1 Sequence Diagram



5.2 Significant Algorithms

5.2.1 Upload Data

1. Bundle data into JSON
2. Send JSON To Server
3. Wait for confirmation...
4. If confirmation timeout reached... prompt user to resend JSON.

5.2.2 Bundle Data

1. Base 64 encode all images
2. Collate data for each waypoint into JSON
3. Collate every waypoint route metadata into JSON

5.2.3 Validate Inputs

1. Check if empty
2. Check if contains "unsanitised" characters
3. Check image sizes

5.2.4 Change Screen

1. Validate Inputs
2. Display next screen

5.3 Significant Data Structures - Android

The main classes in our design are the Route class, and the Waypoint class.

5.3.1 Route

The route class is responsible for storing and processing all the information for one particular route. It does this by storing basic route information, plus a linked list of waypoints. We chose a linked list, as it's size is not fixed, meaning that we are not limited in terms of how many waypoints we can have. We chose a linked list over an arraylist, as an arraylist would be innefficeint and slow, due to the processes involved when adding new items.

5.3.2 Waypoint

The waypoint class is responsible for storing and processing the data for each waypoint along the route. It does this by storing basic route information, plus a linked list of co-ordinates between the current waypoint and the previous waypoint. The timestamp is also generated when the waypoint is constructed. It can also store an optional image, if the user so wishes to add one.

5.3.3 Co-ordinates

The android.location.Location class from the Android API will be used for storing coordinates.

5.4 Significant Data Structures - MySQL Database

To store data about tours a database will be used, this will use MySQL and be the gateway between the android program and the web program. The database shall contain the four tables below with the relevent columns.

5.4.1 List of Walks

Each walk is listed in the "walks" table. There is one record for each walk, and the record has the following characteristics. An ID which is used to order the list of the walks, it will be the primary key for the record and generated by the database. Each walk has a title which represents the name of the walk set by the user and will be of type varchar. The tour also has both short and long descriptions. The short description is a short summary of the walk, where as the long description goes into more detail about the tour. Both of these columns will be varchars. The tour also contains details of how long the walk will take and this will be stored as a float in the database. This will be generated by taking a time for the start of the walk and a time when the walk is finished, then calculating the difference. The final data stored about a walk is the distance and this will also be stored as a float. To create this value the distance between each waypoint will be added.

5.4.2 Location

In this table basic details will be stored about each Waypoint within the walk. The primary key in is the ID and will be generated by the database. This table also uses a foreign key to reference which walk is part of in the walks table, which will be called walkID. Next in the table are two columns for longitude and latitude, both of which are stored as floats. Finally the last column in the table is the timestamp and this will be stored as a float.

5.4.3 Place Description

This table is used to store extra details about each waypoint within a walk. Therefore this table needs an ID as primary key and also a foreign key to link to the location table, which will be called locationID. There is also a column for description which gives a description about the Waypoint. The description will be of type varchar.

5.4.4 Photo Usage

This table is very similar to the place description table, but stores a file path to an image of the location. The table needs to have a primary key (id) for each image as well as a foreign key linked to the location which it is about (placeID). The final column in the table is a varchar which stores the Base64 representation of an image - "photoData".

5.5 Website Interface Specification

The web interface will use all of the following classes to dynamically create a page that will acquire information from a database of routes. The user will also be able to choose from a range of tours provided right next to the map.

5.5.1 Page

The page class is the web page that will hold the map and database interaction. This class will be the way the interface is shown by using css. This class will hold the map class and the database interaction class. The page class will be done in HTML and it will validate to XHTML 1.0 Strict.

5.5.2 Map

The Map class is the class which holds the Google maps API which will be in JavaScript. This class holds functions that are provided from the Google maps API website that initialise the maps and create the pointers on the map. There also other function such as myclick() which records when the user click on the map. Also there is a createMarker() function which will create an info box for each point provided this is were we will provide the information for each place by using the database interaction here to create all the pointers. Below are some functions in JavaScript that the web interface will use with the Google Maps API.

5.5.3 Database Interaction

The database interaction class is where the web page will connect to the databases then It will use JQuery to get all the points on the tour that the user choose. It will present the tours as links to the left of the map which when clicked will connect to that database and load in all the points to the map and load pointer to the right of the map. The map will have information boxes for each point on the map which have had information put in them by using JQuery as the information for each point is save alongside the coordinates. A thumbnail of an image of the location will also be in these information boxes and when clicked on a bigger image will show at the centre of the page. PHP code will be used to read in the databases of tours which will contain each tours the code will create links on the left of the map to bring up a new tour and load in the pointer. The PHP code used will mostly be in the "create pointer" part of the map class where it will create a pointer for each entry in the database which will then create a link on the right of the map to link to each pointer on the map.

5.6 JSON Example

```
{
    "Walk": {
        "walkTitle": "Demo Walk Title",
        "shortDescription": "Demo Short Description",
        "longDescription": "Demo Long Description",
        "walkHours": 42,
        "walkDistance": 42,
        "Locations": [
            {
                "LocationTitle",
                "GPSTraces" : [
                    {
                        "latitude": 42.00000,
                        "longitude": 42.00000
                    }
                ],
                "timestamp": 16:20,
                "locationDescription": "Demo Location Description",
                "Photo": {
                    "photoName": "Photo of Location",
                    "64bitPhoto": "gfkfhjfhkhfctyb7r67tny8b6n756"
                }
            }
        ]
    }
}
```

5.7 HTTP Transaction and Expected Values

The JSON transaction is expected to be completed using the HTTP POST protocol. A check for successful upload will be performed by checking the HTTP response code (success = 200).

6 Android Design Justifications

- We decided to implement Base64 encoding on our images within our application - this allows the developers to directly embed the Base64-encoded image data within the webpage - and it's a simple API call to convert to Base64 within Android.
- We also decided to use JSON within the MIME/HTTP POST request. This would allow us to transfer data between client and server in a much more human-readable (and therefore maintainable) manner. Furthermore, JSON is a much more modern standard than mimicking an HTTP form POST, and just as simple to implement. The team spoke to the customer, and they approved the design change.

7 Web Design Justifications

- We decided to use the Aberystwyth Users web server (http://users.aber.ac.uk/) for hosting the website. This is because it is fast, reliable, well-maintained and stable, as well as running on a solid Linux platform. The server also runs our chosen services - namely PHP and MySQL. We chose those two technologies as PHP is a long-recognised and popular web programming language, and MySQL has an enormous user base and is extremely stable, as well as implementing all of the SQL functions we plan to use.
- Google Maps was chosen for its simple and well-documented API, as well it's flexible JavaScript library. It also has a very well-known and highly attractive interface, which fits in with our interface design plan.

8 REFERENCES

[1] Google. Google Android API Reference.
http://developer.android.com/reference/android/location/Location.html.
Accessed: Dec 05, 2013.

[2] Google. Google Maps API JavaScript Reference.
https://developers.google.com/maps/documentation/javascript/3.exp/reference.
Accessed: Dec 05, 2013.

[3] Hurnherr, T. LaTeXListings Package Info.
http://texblog.org/2008/04/02/include-source-code-in-latex-with-listings/.
Accessed: Dec 03, 2013.

[4] Pavlic, T. Programming Assignment template for LaTeX.
http://www.latextemplates.com/template/programming-coding-assignment.
Accessed: Oct 23, 2013.

[5] Various. LaTeXListings Java Highlighting.
http://tex.stackexchange.com/questions/115467/listings-highlight-java-annotations.
Accessed: Dec 03, 2013.

[6] Various. LaTeXListings JavaScript Highlighting.
http://tex.stackexchange.com/questions/89574/language-option-supported-in-listings.
Accessed: Dec 03, 2013.

9 VERSION HISTORY

Author	Date	Version	Change made	CCF
CCN	03/12/2013	1.0	Initial build of document	n/a
DEC	04/12/2013	1.1	Updated Markdown conversion issues	#33
CCN	04/12/2013	1.2	Corrected spelling and grammar mistakes	#34
CCN	05/12/2013	1.3	Inserted missing sections	#34
CCN	05/12/2013	1.4	Rearranged to reflect order in QA doc	#35
CCN	14/02/2014	1.5	Removed broken document markup	#35
CCN	14/02/2014	1.6	Updated interfaces, diagrams and wording	#36