Group 10 Design Specification

December 6, 2013

SE.10.D3

Version: 1.0
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

1.1 Purpose of This Document

The purpose of the document is to give the client an understanding of how the project is going to be structured and developed. 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. This document is intended to be read and understood by the client.

1.2 Scope

This design specification will separate the project into components which are implemented separately from each other - describing the interactions between components. Conventions used in more technical based documents will not be necessary, as this is a document for the client's understanding.

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 DataEntryActivity

Inherits from AbstractActivity. Contains common code for data entry (text validation, etc).

2.3.3 CreateWaypoint

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

2.3.4 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.

2.3.5 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 page class will make use of the Map and the Database Interaction classes. This class will be written in HTML and CSS, this class will display the map and web interface.

2.4.2 Map

The Map class will be the interface to the Google Maps API; which will be in JavaScript. This class will be used in conjunction with the database interaction to load information into the map in the web interface, along with other details which will be entered into the information box on each location. This class will be implemented using JavaScript and PHP.

2.4.3 Database Interaction

The database interaction class is where the database will be loaded; to 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. This class will implemented with PHP and MySQL.

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 over HTTP.

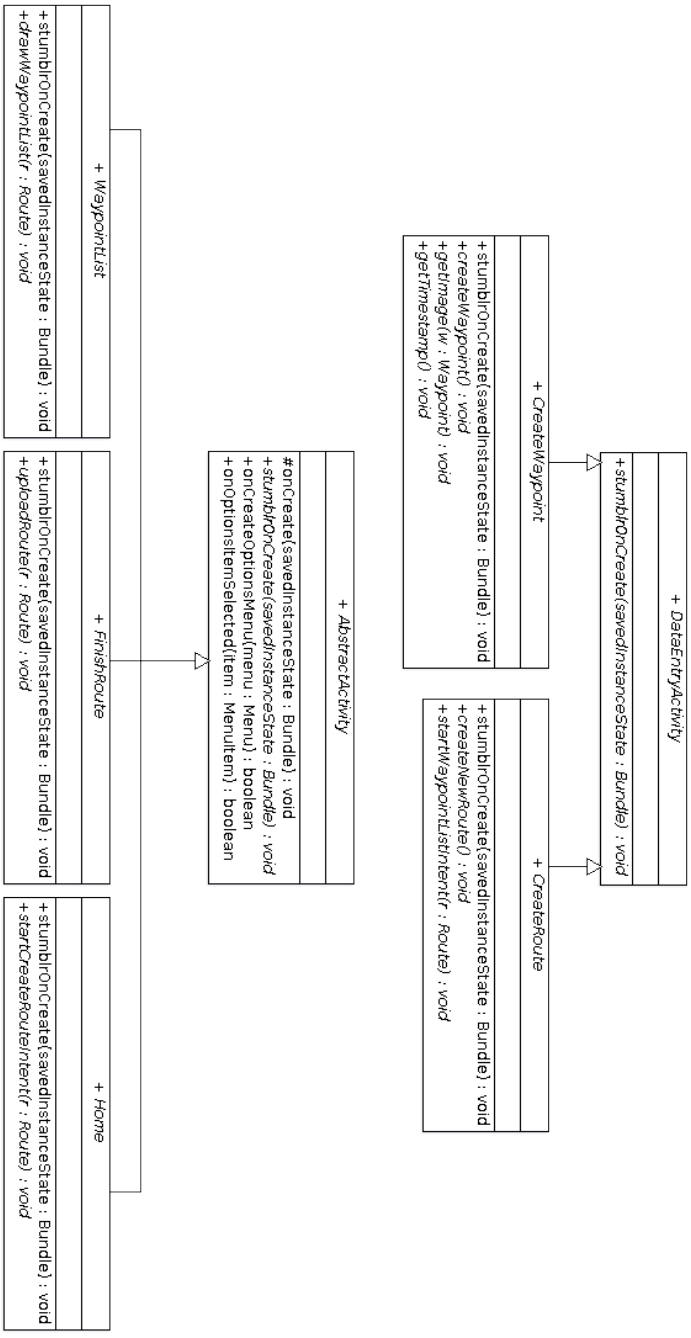
2.6 Mapping Requirements to Classes

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 AbstractActivity, Android class DataEntryActivity and Android class AbstractActivity.
3. FR3; Adding locations to the walking tour Android class CreateWaypoint, Android class DataEntry-Activity and Android class AbstractActivity.
4. FR4; Adding photos to the walks Android class CreateWaypoint, Android class DataEntryActivity 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.
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.

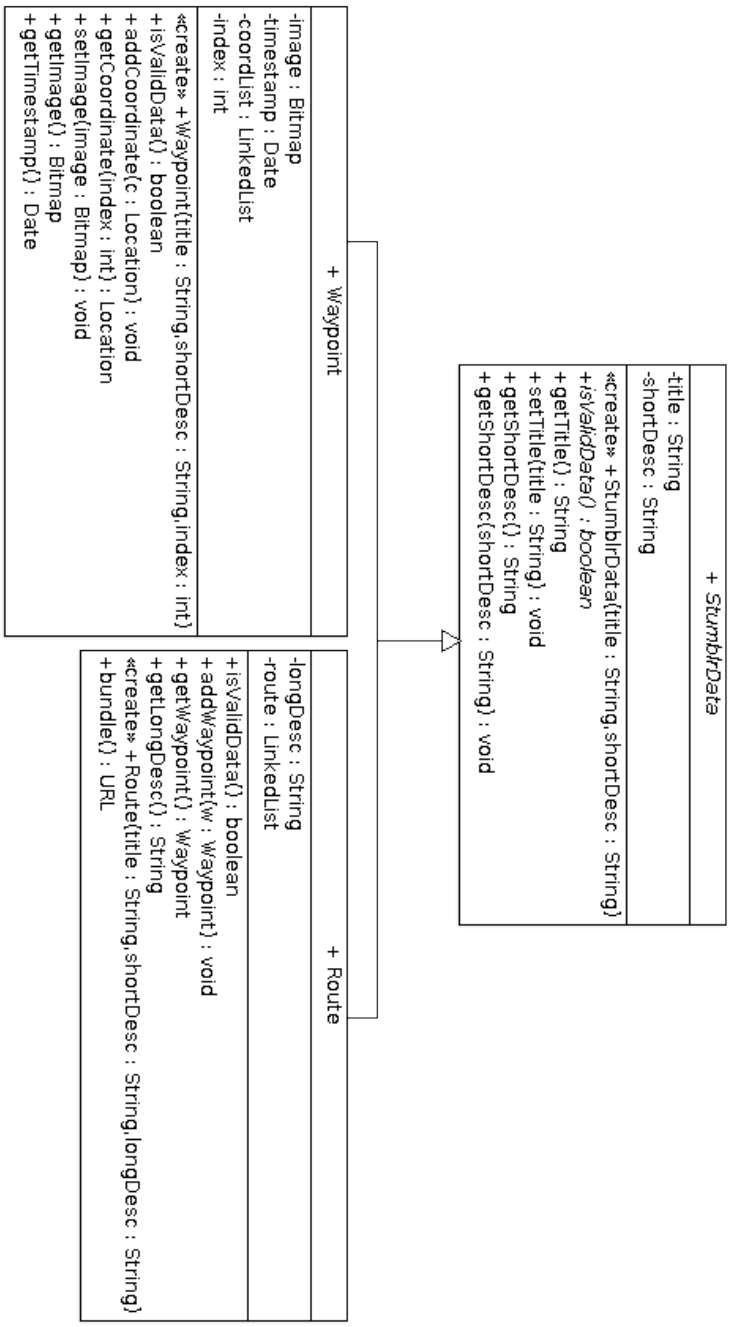
3 DEPENDENCY DESCRIPTION

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.group10.stumblr;
2
3  import android.support.v7.app.ActionBarActivity;
4  import android.support.v4.app.Fragment;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.Menu;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.view.ViewGroup;
11
12 public abstract class AbstractActivity extends ActionBarActivity {
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         stumblrOnCreate(savedInstanceState);
17     }
18
19     public abstract void stumblrOnCreate(Bundle savedInstanceState);
20
21     @Override
22     public boolean onCreateOptionsMenu(Menu menu) {
23
24         // Inflate the menu; this adds items to the action bar if it is
               present.
25         getMenuInflater().inflate(R.menu.main, menu);
26         return true;
27     }
28
29     @Override
30     public boolean onOptionsItemSelected(MenuItem item) {
31         // Handle action bar item clicks here. The action bar will
32         // automatically handle clicks on the Home/Up button, so long
33         // as you specify a parent activity in AndroidManifest.xml.
34         int id = item.getItemId();
35         if (id == R.id.action_settings) {
36             return true;
37         }
38         return super.onOptionsItemSelected(item);
39     }
40
41     /**
42      * A placeholder fragment containing a simple view.
43      */
44     public static class PlaceholderFragment extends Fragment {
45
46         public PlaceholderFragment() {
```

```
47          }

49          @Override
            public View onCreateView(LayoutInflater inflater, ViewGroup
                container,
51                  Bundle savedInstanceState) {
                View rootView = inflater.inflate(R.layout.fragment_abstract,
                    container, false);
53              return rootView;
            }
55      }

57  }
```

4.1.2 DataEntryActivity {abstract}

Listing 2: DataEntryActivity {Abstract}

```
package uk.ac.aber.cs.group10.stumblr;

import android.support.v7.app.ActionBarActivity;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

public abstract class DataEntryActivity extends ActionBarActivity {
    /**
     * Load the activity on creation (using a bundle if one is present)
     * @param savedInstanceState The bundle containing the saved instance
        state.
     */
    public abstract void stumblrOnCreate(Bundle savedInstanceState);
}
```

4.1.3 Home

Listing 3: Home

```
package uk.ac.aber.cs.group10.stumblr;

import android.os.Bundle;

import uk.ac.aber.cs.group10.stumblr.data.Route;

public abstract 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) {
        // Called by super().onCreate
        setContentView(R.layout.activity_abstract);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.container, new PlaceholderFragment())
                    .commit();
        }
    }

    /**
     * Begin the Route entry activity
     */
    public abstract void startCreateRouteIntent(Route r);
}
```

4.1.4 CreateRoute

Listing 4: CreateRoute

```
1  package uk.ac.aber.cs.group10.stumblr;
2
3  import android.os.Bundle;
4
5  import uk.ac.aber.cs.group10.stumblr.data.Route;
6
7  public abstract class CreateRoute extends DataEntryActivity {
8      /**
9       * Loads the activity on creation (using a bundle if one is present)
10      * @param savedInstanceState The bundle containing the saved instance
            state.
11      */
12     @Override
13     public void stumblrOnCreate(Bundle savedInstanceState) {
14         // Called by super().onCreate
15         setContentView(R.layout.activity_abstract);
16
17         if (savedInstanceState == null) {
18             getSupportFragmentManager().beginTransaction()
19                     .add(R.id.container, new AbstractActivity.
                           PlaceholderFragment())
20                     .commit();
21         }
22     }
23
24
25     /**
26      * Create a new Route object (using the information entered by the
            user)
27      */
28     public abstract void createNewRoute();
29
30     /**
31      * Start the WaypointList activity (list the current Route).
32      */
33     public abstract void startWaypointListIntent(Route r);
34 }
```

4.1.5 CreateWaypoint

Listing 5: CreateWaypoint

```
package uk.ac.aber.cs.group10.stumblr;

import android.os.Bundle;

import uk.ac.aber.cs.group10.stumblr.data.Route;
import uk.ac.aber.cs.group10.stumblr.data.Waypoint;

public abstract class CreateWaypoint extends DataEntry {
    /**
     * 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) {
        // Called by super().onCreate
        setContentView(R.layout.activity_abstract);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.container, new AbstractActivity.
                       PlaceholderFragment())
                    .commit();
        }
    }

    /**
     * Create a new Waypoint based on user input.
     */
    public abstract void createWaypoint();

    /**
     * Obtain a photo from user and add it to current Waypoint.
     */
    public abstract void getImage(Waypoint w);

    /**
     * Set a timestamp on the current Waypoint.
     */
    public abstract void getTimestamp();

    /**
     * Obtain coordinates from Android system and add to current Waypoint.
     */
}
```

4.1.6 WaypointList

Listing 6: WaypointList

```
package uk.ac.aber.cs.group10.stumblr;

import android.os.Bundle;

import uk.ac.aber.cs.group10.stumblr.data.Route;

public abstract class WaypointList 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) {
        // Called by super().onCreate
        setContentView(R.layout.activity_abstract);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.container, new AbstractActivity.
                        PlaceholderFragment())
                    .commit();
        }
    }

    /**
     * Renders Waypoint list on screen.
     * @param r The Route object containing Waypoints to render.
     */
    public abstract void drawWaypointList(Route r);
}
```

4.1.7 FinishRoute

Listing 7: FinishRoute

```
package uk.ac.aber.cs.group10.stumblr;

import android.os.Bundle;

import uk.ac.aber.cs.group10.stumblr.data.Route;

public abstract class FinishRoute 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) {
        // Called by super().onCreate
        setContentView(R.layout.activity_abstract);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.container, new PlaceholderFragment())
                    .commit();
        }
    }

    /**
     * Will package and upload the current Route object.
     */
    public abstract void uploadRoute(Route r);
}
```

4.2 Data Classes

4.2.1 StumblrData {abstract}

Listing 8: StumblrData

```
1  package uk.ac.aber.cs.group10.stumblr.data;
2
3  /**
4   * Abstract class containing basic common structure for all Stumblr data
       formats.
5   */
6  public abstract class StumblrData {
7      /**
8       * The title of the given piece of data. This can be extended to
           Waypoints, Routes
9       * and any other piece of relevant data inside the structure of
           Stumblr.
10      */
11     private String title;
12
13     /**
14      * The short description pertaining to the given piece of data. This
           will also be
15      * extended to other component classes of StumblrData
16      */
17     private String shortDesc;
18
19     /**
20      * @param title The title to set.
21      * @param shortDesc The short description to set.
22      */
23     public StumblrData(String title, String shortDesc) {
24         this.title = title;
25         this.shortDesc = shortDesc;
26     }
27
28     /**
29      * Checks the StumblrData item for validity. Returns a boolean. (true
           = valid)
30      * @return Whether the data is valid or not. (true = valid)
31      *
32      * MUST be implemented in any subclasses.
33      */
34     public abstract boolean isValidData();
35
36     /**
37      * Returns the title.
38      * @return this.title
39      */
40     public String getTitle() {
41         return this.title;
42     }
43
44     /**
```

```
45          * Sets the current title.
            * @param title The title to set.
47          */
        public void setTitle(String title) {
49          this.title = title;
        }
51
        /**
53          * Returns the short description
            * @return this.shortDesc
55          */
        public String getShortDesc() {
57          return this.getShortDesc();
        }
59
        /**
61          * Sets the current short description.
            * @param shortDesc The short description to set.
63          */
        public void getShortDesc(String shortDesc) {
65          this.shortDesc = shortDesc;
        }
67  }
```

4.2.2 Route

Listing 9: Route

```
1  package uk.ac.aber.cs.group10.stumblr.data;
2
3  import java.net.URL;
   import java.util.LinkedList;
5
   /**
7   * Created by charles on 29/11/13.
    */
9  public class Route extends StumblrData {
       /**
11      * A slightly longer description of the contents of the route. Set by
            the user when
        * they create a Route.
13      */
       private String longDesc;
15
       /**
17      * A LinkedList of Waypoint objects that the Route comprises of.
        */
19     private LinkedList<Waypoint> route;
21     /**
        * Checks if the data in the Route is valid or not, and returns a
            boolean.
23      * @return If the data is valid or not. (true = valid)
        */
25     public boolean isValidData() {
           return false;
27     }
29     /**
        * Adds a Waypoint to the tail of the Route LinkedList
31      * @param w The waypoint to add
        */
33     public void addWaypoint(Waypoint w) {
           this.route.addLast(w);
35     }
37     /**
        * Returns the last Waypoint to the LinkedList.
39      * @return The last Waypoint in the Route.
        */
41     public Waypoint getWaypoint() {
           return this.route.getLast();
43     }
45     public String getLongDesc() {
           return this.longDesc;
47     }
49     /**
```

```
     * Constructor for Route.
51   * @param title The title of the Route object.
     * @param shortDesc A short description of the Route.
53   * @param longDesc A longer description of the Route.
     */
55  public Route(String title, String shortDesc, String longDesc) {
        super(title, shortDesc);
57
    }
59
    /**
61   * To be implemented. Will return a URL containing filesystem location
         of bundled Route file
     * (ready for upload to server)
63   * @return The URL of the bundle
     */
65  public URL bundle() {
        return null;
67  }
}
```

4.2.3 Waypoint

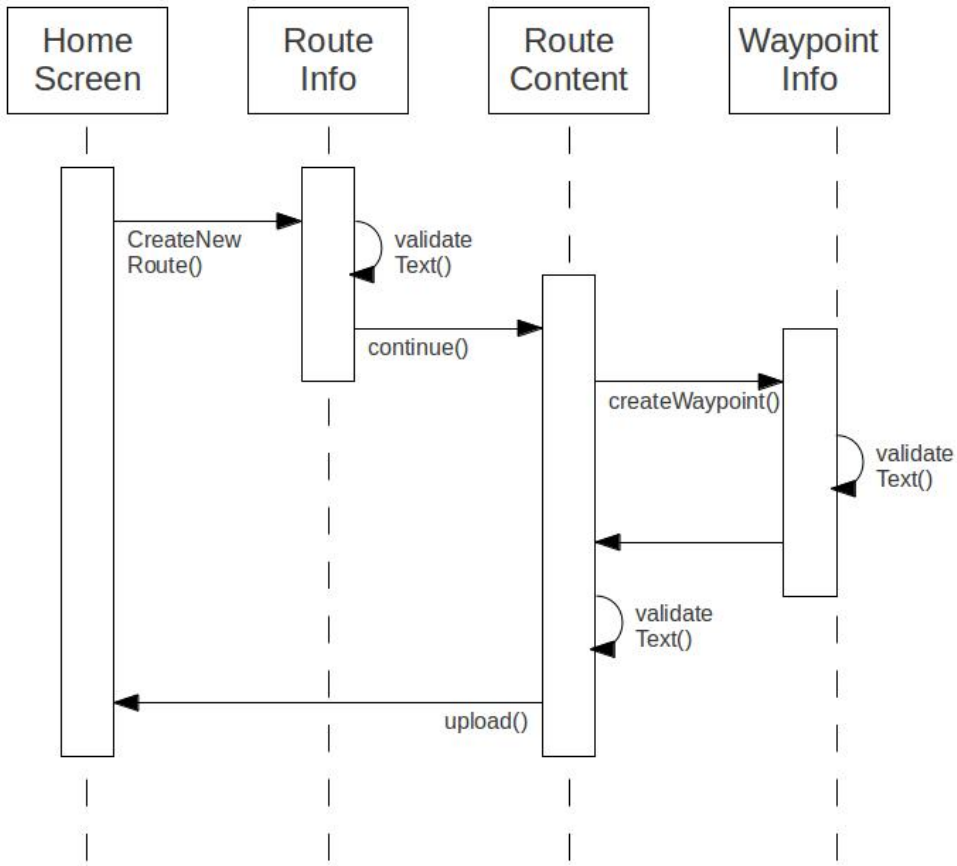
Listing 10: Waypoint

```
package uk.ac.aber.cs.group10.stumblr.data;

import android.graphics.Bitmap;
import android.location.Location;

import java.util.Date;
import java.util.LinkedList;

/**
 * Created by charles on 29/11/13.
 */
public class Waypoint extends StumblrData {
    /* title and shortDesc are declared in StumblrData and are accessed
       through get/set methods */

    /**
     * Image contained within Waypoint.
     */
    private Bitmap image;

    /**
     * Time at which Waypoint was created.
     */
    private Date timestamp;

    /**
     * List of coordinates (from the last location to the current location
        )
     */
    private LinkedList<Location> coordList;

    private int index;

    /**
     * Constructor for a Waypoint object.
     * @param title Title of the waypoint.
     * @param shortDesc A short description.
     */
    public Waypoint(String title, String shortDesc, int index) {
        /* Calls superclass constructor */
        super(title, shortDesc);

        /* Sets index variable (used for locating position in array */
        this.index = index;

        /* Uses Android system to get time. */
        this.timestamp = new Date();
        /* Initialise LinkedList */
        this.coordList = new LinkedList<Location>();
    }
```

```
50      /**
        * To be implemented.
52      * @return Validity of data (true = valid)
        */
54      public boolean isValidData() {
            return false;
56      }

58      /**
        * Add a coordinate to the list.
60      * @param c The location to be added to the tail of the LinkedList.
        */
62      public void addCoordinate(Location c) {
            this.coordList.addLast(c);
64      }

66      /**
        * Returns coordinate instance at specified index.
68      * @param index Specified index of coordinate.
        * @return The specified coordinate.
70      */
        public Location getCoordinate(int index) {
72          return this.coordList.get(index);
        }
74
        /**
76      * Sets the current image.
        * @param image The image to add to the Waypoint.
78      */
        public void setImage(Bitmap image) {
80          this.image = image;
        }
82
        /**
84      * Returns current image.
        * @return The current image that the Waypoint has,
86      */
        public Bitmap getImage() {
88          return this.image;
        }
90
        /**
92      * Returns current timestamp.
        * @return The current timestamp.
94      */
        public Date getTimestamp() {
96          return this.timestamp;
        }
98

100 }
```

5 DETAILED DESIGN

5.1 Sequence Diagram



5.2 Significant Algorithms

5.2.1 Upload Data

1. Bundle data
2. Cache Bundle
3. Send Bundle To Server
4. Display Progress bar in android drop down menu
5. Wait for confirmation...
6. If confirmation timeout reached... resend bundle.

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.

5.5.4 Google Maps API JavaScript Example

Listing 11: Google Maps API Javascript Example

```
<script type="text/javascript">
  var side_bar_html = "";
  var gmarkers = [];
  var map = null;


//initializes the Google Map.
function initialize() {
// create the map
    var myOptions = {
        zoom: 15,
        //sets the map to Aberystwyth
        center: new google.maps.LatLng(52.4140,-4.0810),
        mapTypeControl: true,
        mapTypeControlOptions: {style: google.maps.MapTypeControlStyle.
           DROPDOWN_MENU},
        navigationControl: true,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }

map = new google.maps.Map(document.getElementById("map_canvas"), myOptions
   );

    google.maps.event.addListener(map, 'click', function() {
        infowindow.close();
    });
```

```
26      //creates pointers - this will be used with the database interaction
            class
        var point = new google.maps.LatLng(,);
28      var marker = createMarker(point,"NAME","DESCRIPTION")

30      document.getElementById("side_bar").innerHTML = side_bar_html;
    }
32
    //creates the map and set the size of the map.
34  var infowindow = new google.maps.InfoWindow(
    {
36    size: new google.maps.Size(200,75)
    });
38


40  function myclick(i) {
        google.maps.event.trigger(gmarkers[i], "click");
42  }

44  function createMarker(latlng, name, html) {
      var contentString = html;
46    var marker = new google.maps.Marker({
          position: latlng,
48        map: map,
          zIndex: Math.round(latlng.lat()*-100000)<<5
50        });

52    google.maps.event.addListener(marker, 'click', function() {
          infowindow.setContent(contentString);
54        infowindow.open(map,marker);
          });
56
      gmarkers.push(marker);
58    side_bar_html += '<a href="javascript:myclick(' + (gmarkers.length-1) +
          ')">' + name + '<\/a><br>';
    }
60  </script>
```

5.6 JSON Example

```
{
    "Walk": {
        "walkTitle": "Road Rummer",
        "shortDescription": "meep meep!",
        "longDescription": "meep meep meep meep!",
        "walkHours": 42,
        "walkDistance": 42,
        "Locations": [
            {
                "LocationTitle",
                "GPSTraces" : [
                    {
                        "latitude": 42.00000,
                        "longitude": 42.00000
                    }
                ],
                "timestamp": 16:20,
                "locationDescription": "This is a bar, can you say bar?",
                "Photo": {
                    "photoName": "Photo of Rummers",
                    "64bitPhoto": "gfkfhjfhkhfctyb7r67tny8b6n756"
                }
            }
        ]
    }
}
```

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 statements 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
CCN	03/12/2013	1.0	Initial build of document
DEC	04/12/2013	1.1	Updated Markdown conversion issues
CCN	04/12/2013	1.2	Corrected spelling and grammar mistakes
CCN	05/12/2013	1.3	Inserted missing sections
CCN	05/12/2013	1.4	Rearranged to reflect order in QA doc