Group Project 05

Design Specification

|  |  |  |
| --- | --- | --- |
| Authors: | bmo; srr11; hac22; wjl3; wia2; njv1; spc3 | Department of Computer Science |
| Config Ref: | SE\_05\_DS\_01 | Aberystwyth University |
| Date: | 06/12/13 | Aberystwyth, Ceredigion, SY23 3DB |
| Version: | 1.2 | Copyright © Group 05 |
| Status: | Release | Aberystwyth University 2013 |

Table of Contents

[1. Introduction 4](#_Toc378774921)

[1.1. Purpose of this document 4](#_Toc378774922)

[1.2. Scope 4](#_Toc378774923)

[1.3. Objectives 4](#_Toc378774924)

[2. Decomposition Description 5](#_Toc378774925)

[2.1. Programs in system 5](#_Toc378774926)

[2.2. Significant classes in the Android program 5](#_Toc378774927)

[2.2.1. Main Activity 5](#_Toc378774928)

[2.2.2. Tour class 5](#_Toc378774929)

[2.2.3. Location class 5](#_Toc378774930)

[2.2.4. Communication class 5](#_Toc378774931)

[2.2.5. Walk activity class 6](#_Toc378774932)

[2.2.6. Link interface 6](#_Toc378774933)

[2.3. Web program components 6](#_Toc378774934)

[2.3.1. Database 6](#_Toc378774935)

[2.3.2. Website 6](#_Toc378774936)

[2.4. Table mapping requirements 7](#_Toc378774937)

[3. Dependency Description 8](#_Toc378774938)

[3.1. UML Component Diagram for the Android application 8](#_Toc378774939)

[3.2. UML Component Diagram for the Database 9](#_Toc378774940)

[4. Interface description 10](#_Toc378774941)

[4.1. Android Class Diagram 10](#_Toc378774942)

[4.2. Main activity class 11](#_Toc378774943)

[4.3. Tour class 11](#_Toc378774944)

[4.4. Location class 13](#_Toc378774945)

[4.5. Key location class 14](#_Toc378774946)

[4.6. Communication class 14](#_Toc378774947)

[4.7. Link interface 15](#_Toc378774948)

[4.8. Walk activity class 16](#_Toc378774949)

[4.9 About Activity 17](#_Toc378774950)

[5. Detail Design 18](#_Toc378774951)

[5.1. Sequence Diagrams 18](#_Toc378774952)

[5.1.1. Tour creation sequence diagram 18](#_Toc378774953)

[5.1.2. Add key location sequence diagram 19](#_Toc378774954)

[5.1.3. Data transfer sequence diagram 19](#_Toc378774955)

[5.1.4. Server side sequence diagram 20](#_Toc378774956)

[5.2. Algorithm Description 21](#_Toc378774957)

[5.2.1. Server Side Data Delivery Algorithm 21](#_Toc378774958)

[5.2.2. Android side algorithms 21](#_Toc378774959)

[5.3. Data Structures 22](#_Toc378774960)

[5.3.1. Android data structures 22](#_Toc378774961)

[5.3.2. Server side data structures 22](#_Toc378774962)

[5.4. Entity Relationship Diagram 23](#_Toc378774963)

[**6.** APPENDIX **A** 24](#_Toc378774964)

[**7.** APPENDIX **B** 25](#_Toc378774965)

[8. References 27](#_Toc378774966)

[9. Document History 27](#_Toc378774967)

# Introduction

## Purpose of this document

The purpose of this document is to describe the outline design for the Walking Tour Creator system, consisting of an Android application, database and website taking into account the details of the group project requirements and group project quality assurance.

## Scope

This document includes detailed description of:

* The classes used in the Android application;
* The methods used in each class;
* Sequence diagrams;
* Interaction between the application and the database;
* Database design and data handling;
* Website design.

This document should be read after familiarisation with the Project Management Plan [1] and Test Specification [2]. The specifications listed in General Document Standards [3], Design Specification Standards [4] and Java Coding Standards [5] have been followed in the process of constructing this Design Specification.

## Objectives

The objective of this document is:

* To describe the main components of the Walking Tour Application;
* To describe the main components of the database and website;
* To depict the dependencies between the components.

# Decomposition Description

## Programs in system

The Walking Tour System consists of two main components:

* The Android application;
* The Website and database;

The Android application will allow the user to create a walking tour. This tour will consist of automatically added locations through GPS along the walk as well as separate points of interest the user has added on their own along with a description and possibly photos. When the user is done, the walk can be uploaded to the online database in the form of a MIME message containing the information on the walk formatted as a JSON file. This message is sent through HTTP POST, intercepted by the PHP on the site and gets stored in the database. The Website allows the user to view walks currently stored in the database by querying it and displaying the walks using the Mapping API.

## Significant classes in the Android program

### Main Activity

This is the launcher activity for the app, when the app first starts this is  the screen the user is presented with.

### Walk Activity

This is the main part of the app, this where the user builds their tour.  From here they can:

* Add/remove points of interest.
* Add/remove photos to/from a point of interest.
* Cancel their walk.
* Save their walk for later viewing.
* They are also able to adjust the sample rate for non-key locations to save on how much data they transmit to the server.

### About Activity

This is just a credits screen. It mentions who is on the team, the app  name and version number. There are also some supporting UI classes for dialogs.

### Tour Save

### WTC Tour

### WTC Location

### WTC KeyLocation

## Web program components

### Database

The Server side of the system consists of a database, holding the information on the tours and a website that displays the tours on an interactive map to the user.

All the data is packed into a MIME message containing all the information about the tour formatted as a JSON file. Appendix A defines the format of the MIME message; Appendix B defines the format of the JSON file. The message is transmitted from the android device in a HTTP POST request. The value will be paired using the key “message”. The data is accessed in PHP using the $\_POST [‘message’] handle. The request is made to the file upload.php which is stored in the root of the website. All requests will be recorded in by upload.php in a file called log.txt in the root of the site as per the testing strategy**.**

The data set will need to contain a title for the tour, a short description of the tour and a long description, a collection of GPS coordinates for the route, a collection of locations associated with GPS coordinates on the route including images and description, the total time of the route and the total distance of the route.

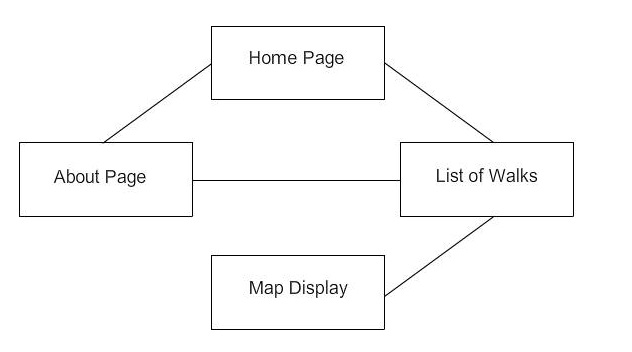
The Database will have the following structure:

|  |  |
| --- | --- |
| Table Name | Table Description |
| **ListOfWalks** | A list of walks/ tours, which the program will display. |
| **Location** | A list of geographical locations referencing a record in the tour table, describing the route of the tour as a sequence of locations. |
| **PlaceDescription** | A list of points of interest along tours referencing a location. |
| **PhotoUsage** | A list of photographs referencing a point of interest |

### Website

The Website consists of 4 main pages:

|  |  |
| --- | --- |
| Page Name | Page Description |
| **Home page** | The index page containing general information. |
| **List of walks page** | Displays a list of walks available to view from the database. |
| **Contacts page** | Information on how to contact the group. |
| **About page** | Other info for the site such as version and links. |



The website will contain the following pages:

On the Home page there will be a little bit of information about the WTC project and a few of the most common routes. The layout of the home page has a content which will hold the most used routes. There will be a map on that screen to display one of the selected walks immediately.

The list of walks page contains a list of all walks in the database that the user can select to view on the map.

The map page will have the mapping API and will take the information on locations for a walk from the database and display them. Locations added by the user will be indicated by pins, which on click will display additional information the user has given and a list of photos that can be viewed.

The Contact page is so people can get in touch and contact us via a form which will be written in either JavaScript or PHP which will allow people to send us feedback and suggestions on how to improve on the system.

The About page will contain general information for the site such as creators, copyright and other legal matters and site version. The names of the developers and a contact e-mail address will be present so we can be contacted with recommendations etc.

All of the pages are accessible via a Navigational Bar under the header with buttons to each of the pages on the website, allowing for a dynamic experience and feel to the site as the buttons change upon behaviour of highlighted, clicked and also passive. The map will only be displayed when a walk is selected to be displayed.

## Table mapping requirements



# Dependency Description

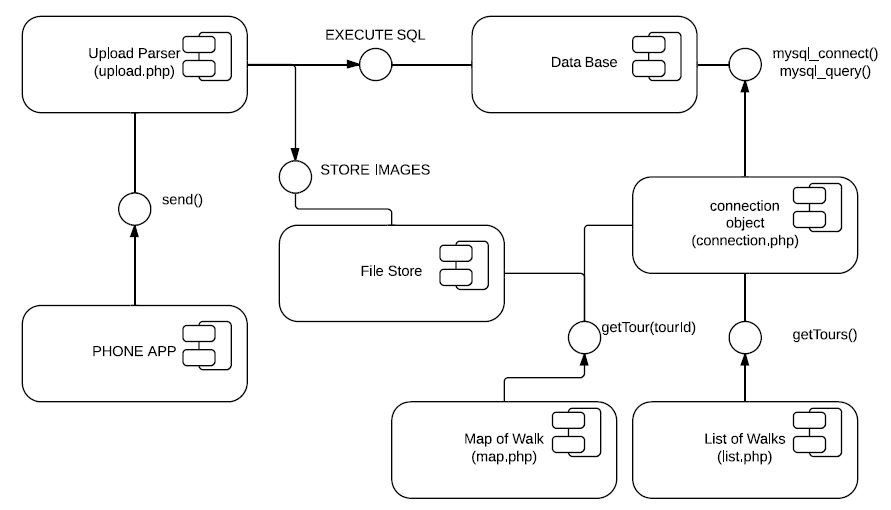
## UML Component Diagram for the Android application

The diagram depicts the different components, the Android application consists of and how they interact with each other. The User Interface component can interact with the Tour, Location and Communication components through the respective methods depicted in the diagram.



## UML Component Diagram for the Database

The diagram depicts the different components, the Server side application consists of and how they interact with each other. After the data is received from the phone application it is processed by the Upload Parsed component and sent to the Database component. From there depending on what the user wants to see, either the List of walks or the Map component is called.



# Interface description

This section contains all of the classes in the Android application and their class skeletons, depicting the methods that are to be implemented in them along with the appropriate JavaDoc commenting for clarification.

## Android Class Diagram

cshome:Downloads:WTC+classes (1).pdfThe UML class diagram depicts all the significant classes the application consists of. The default Android classes such as Activity are extended by some the program uses like MainActivity etc. but are omitted from the class diagram.

## Main Activity class

*/\**

*\* @(#) MainActivity.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **MainActivity** **extends** Activity **implements** OnClickListener**,** LocationListener **{**

*/\*\**

*\* Creates this activity and sets its layout. It also initializes the buttons and sets*

*\* their listeners.*

*\*/*

@Override

**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

**}**

@Override

**protected** **void** **onStop(){**

**}**

*/\*\**

*\* Dictates what what happens when the user clicks a button.*

*\*/*

@Override

**public** **void** **onClick(**View view**)** **{**

**}**

@Override

**public** **void** **onLocationChanged(**Location location**)** **{**

**}**

@Override

**public** **void** **onProviderDisabled(**String provider**)** **{**

**}**

@Override

**public** **void** **onProviderEnabled(**String provider**)** **{**

**}**

@Override

**public** **void** **onStatusChanged(**String provider**,** **int** status**,** Bundle extras**)** **{**

**}**

**}**

## About Activity class

*/\**

*\* @(#) AboutActivity.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **AboutActivity** **extends** Activity **{**

*/\*\**

*\* Creates this activity and its layout.*

*\*/*

@Override

**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

**}**

## Walk Activity class

*/\**

*\* @(#) WalkActivity.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **WalkActivity** **extends** Activity **implements** ConnectionCallbacks**,** OnConnectionFailedListener**,**

LocationListener**,** OnMarkerClickListener**,** WTCDialogCallbacks**{**

@Override

**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

**}**

@Override

**protected** **void** **onStart()** **{**

**}**

@Override

**protected** **void** **onPause()** **{**

**}**

@Override

**protected** **void** **onResume()** **{**

**}**

@Override

**protected** **void** **onStop()** **{**

**}**

     @Override

**protected** **void** **onActivityResult(int** requestCode**,** **int** resultCode**,** Intent data**)** **{**

**}**

@Override

**public** **void** **onLocationChanged(**Location location**)** **{**

**}**

@Override

**public** **void** **onConnectionFailed(**ConnectionResult result**)** **{**

**}**

@Override

**public** **void** **onConnected(**Bundle bundle**)** **{**

**}**

@Override

**public** **void** **onDisconnected()** **{**

**}**

*/\*\**

*\* Sets the selected marker.*

*\**

*\* Called when the user clicks on a map marker.*

*\*/*

@Override

**public** **boolean** **onMarkerClick(**Marker marker**)** **{**

**}**

*/\*\**

*\* Creates the action bar where all the buttons are.*

*\*/*

@Override

**public** **boolean** **onCreateOptionsMenu(**Menu menu**)** **{**

**}**

*/\*\**

*\* Dictates what happens when the user clicks a button on the options menu.*

*\*/*

@Override

**public** **boolean** **onOptionsItemSelected(**MenuItem item**)** **{**

**}**

*/\*\**

*\* Used to add a new key location to the tour.*

*\*/*

@Override

**public** **void** **onPositiveSelection(**DialogFragment fragment**)** **{**

**}**

*/\*\**

*\* Used to cancel the tour if the user selects 'OK' on the dialog.'*

*\*/*

@Override

**public** **void** **onCancelWalkSelection(**DialogFragment fragment**)** **{**

**}**

@Override

**public** **void** **onFinishWalkSelection(**DialogFragment fragment**)** **{**

**}**

*/\*\**

*\* Adds a location to the tour.*

*\*/*

**public** **void** **addLocation()** **{**

**}**

*/\*\**

*\* Adds a key location to the walk.*

*\*/*

**public** **void** **addKeyLocation(**WTCKeyLocation keyLoc**)** **{**

**}**

*/\*\**

*\* Removes a location from the tour.*

*\**

*\* @param marker the marker being removed from the map; also used to identify which location to remove.*

*\*/*

**public** **void** **removeLocation(**Marker marker**)** **{**

**}**

**public** **void** **addPhoto()** **{**

**}**

*/\**

*\* The following method was taken from the Android Developers Documents*

*\**

*\* URL: http://developer.android.com/training/camera/photobasics.html*

*\*/*

**private** File **createImageFile()** **throws** IOException **{**

**}**

**public** **void** **removePhoto()** **{**

**}**

*/\*\**

*\* Saves the tour to the server.*

*\*/*

**private** **void** **saveToServer()** **{**

**}**

*/\*\**

*\* Stops the recording of the and deletes the data.*

*\*/*

**public** **void** **cancelWalk()** **{**

**}**

*/\*\**

*\* Adds a stopping location to the tour and stops the recording.*

*\*/*

**public** **void** **finishWalk()** **{**

**}**

*/\*\**

*\* @param tour the new value for this.tour.*

*\*/*

**public** **void** **setTour(**WTCTour tour**)** **{**

**}**

*/\*\**

*\* @return the tour.*

*\*/*

**public** WTCTour **getTour()** **{**

**}**

*/\*\**

*\* Sets a new value for this.sampleRate.*

*\**

*\* @param sampleRate the new value for this.sampleRate.*

*\*/*

**public** **void** **setSameRate(long** sampleRate**)** **{**

**}**

*/\*\**

*\* @return the sample rate.*

*\*/*

**public** **long** **getSampleRate()** **{**

**}**

**}**

## WTC DialogCallbacks interface

*/\**

*\* @(#) WTCDialogCallbacks.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **interface** **WTCDialogCallbacks** **{**

*/\*\**

*\* This method is used when the user clicks the positive button on a dialog.*

*\**

*\* @param fragment the DialogFragment that made this callback.*

*\*/*

**public** **void** **onPositiveSelection(**DialogFragment fragment**);**

*/\*\**

*\* This method is to be called when the user clicks the positive button*

*\* on the CancelWalkFragment.*

*\**

*\* @param fragment the DialogFragment that made this callback.*

*\*/*

**public** **void** **onCancelWalkSelection(**DialogFragment fragment**);**

*/\*\**

*\* This method is to be called when the user clicks the positvite button*

*\* on the FinishWalkFragment.*

*\**

*\* @param fragment the DialogFragment that made this callback.*

*\*/*

**public** **void** **onFinishWalkSelection(**DialogFragment fragment**);**

**}**

## TourSave class

*/\**

*\* @(#) TourSave.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **TourSave** **{**

*/\*\**

*\* Constructs a TourSave object with a file path to where the tour data will be saved.*

*\**

*\* @param filePath the file path for where the tour will saved on the device.*

*\*/*

**public** **TourSave(**String filePath**){**

**}**

*/\*\**

*\* Serializes the tour data out to file.*

*\**

*\* @param tour the tour being saved.*

*\* @return a message to say whether the save was successful or not*

*\*/*

**public** String **saveTour(**WTCTour tour**)** **{**

**}**

*/\*\**

*\* Deserializes a tour data file.*

*\**

*\* @return the tour loaded in from file*

*\*/*

**public** WTCTour **loadTour(){**

**}**

**}**

## WTCTour class

*/\**

*\* @(#) WTCTour.java Version 1.0*

*\**

*\* Copyright(c) Group 5 @Aberystwyth University*

*\* All rights reserved*

*\*/*

**public** **class** **WTCTour** **implements** Serializable**{**

*/\*\**

*\* Constructs a blank tour.*

*\*/*

**public** **WTCTour(){**

**}**

*/\*\**

*\* Constructs a new tour with a specified name.*

*\**

*\* @param tourName the name of the tour.*

*\*/*

**public** **WTCTour(**String tourName**){**

**}**

*/\*\**

*\* Constructs a new tour with a specified name and short description.*

*\**

*\* @param tourName the name of the tour.*

*\* @param shortDesc a short description for the tour.*

*\*/*

**public** **WTCTour(**String tourName**,** String shortDesc**){**

**}**

*/\*\**

*\* Constructs a new tour with all necessary details.*

*\**

*\* @param tourName the name of the tour.*

*\* @param shortDesc a short description for the tour.*

*\* @param longDesc a long description for the tour.*

*\*/*

**public** **WTCTour(**String tourName**,** String shortDesc**,** String longDesc**){**

**}**

*/\*\**

*\* Adds a location to the tour.*

*\**

*\* @param location the location being added to the tour.*

*\*/*

**public** **void** **addLocation(**WTCLocation location**){**

**}**

*/\*\**

*\* Removes a location from the tour.*

*\**

*\* @param index the index of the location being removed.*

*\*/*

**public** **void** **removeLocation(int** index**){**

**}**

*/\*\**

*\* Takes the data stored this object and converts it to a JSON String.*

*\**

*\* @return the JSON String.*

*\*/*

**public** String **toJSON(){**

**}**

*/\*\**

*\* Sets a new value for this.tourName.*

*\**

*\* @param tourName the new value for this.tourName.*

*\*/*

**public** **void** **setTourName(**String tourName**){**

**}**

*/\*\**

*\* @return the name of the tour.*

*\*/*

**public** String **getTourName(){**

**}**

*/\*\**

*\* Sets a new value for this.shortDesc.*

*\**

*\* @param shortDesc the new value for this.shortDesc.*

*\*/*

**public** **void** **setShortDesc(**String shortDesc**){**

**}**

*/\*\**

*\* @return the short description for the tour.*

*\*/*

**public** String **getShortDesc(){**

**}**

*/\*\**

*\* Sets a new value for this.longDesc.*

*\**

*\* @param longDesc the new value for this.longDesc.*

*\*/*

**public** **void** **setLongDesc(**String longDesc**){**

**}**

*/\*\**

*\* @return the long description for the tour.*

*\*/*

**public** String **getLongDesc(){**

**}**

*/\*\**

*\* Sets the list of locations in the tour.*

*\**

*\* @param locations the new value for this.locations.*

*\*/*

**public** **void** **setLocations(**LinkedList**<**WTCLocation**>** locations**){**

**}**

*/\*\**

*\* @return the list of locations in the tour.*

*\*/*

**public** LinkedList**<**WTCLocation**>** **getLocations(){**

**}**

*/\*\**

*\* A shortcut method to get the size of the list of locations*

*\**

*\* @return the size of the list of locations in the tour*

*\*/*

**public** **int** **getLocationsSize(){**

**}**

*/\*\**

*\* Adjusts the timestamps for each location in the tour to be*

*\* minutes from the start of the tour.*

*\*/*

**public** **void** **fixTime(){**

**}**

*/\*\**

*\* Calculate a rough estimate for the distance of the tour*

*\*/*

**private** **double** **calcDist(){**

**}**

**private** **static** **double** **GreatCircle(**WTCLocation loc1**,** WTCLocation loc2**){**

**}**

**}**

## WTCLocation class

*/\**

*\* @(#)WTCLocation.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **WTCLocation** **implements** Serializable**{**

*/\*\**

*\* Constructs a blank location.*

*\*/*

**public** **WTCLocation(){**

**}**

*/\*\**

*\* Constructs a location object a specified longitude and latitude.*

*\**

*\* @param longitude the longitude of the location.*

*\* @param latitude the latitude of the location.*

*\*/*

**public** **WTCLocation(double** longitude**,** **double** latitude**){**

**}**

*/\*\**

*\* Sets a new value for this.longitude.*

*\**

*\* @param longitude the new value for this.longitude.*

*\*/*

**public** **void** **setLongitude(double** longitude**){**

**}**

*/\*\**

*\* @return the longitude of the location.*

*\*/*

**public** **double** **getLongitude(){**

**}**

*/\*\**

*\* Sets a new value for this.latitude.*

*\**

*\* @param latitude the new value for this.latitude.*

*\*/*

**public** **void** **setLatitude(double** latitude**){**

**}**

*/\*\**

*\* @return the latitude for the location.*

*\*/*

**public** **double** **getLatitude(){**

**}**

*/\*\**

*\* Sets a new value fort this.timeStamp.*

*\**

*\* The time stamp is set completely by the method and therefore takes no parameters.*

*\*/*

**public** **void** **setTimeStamp(long** newTime**){**

**}**

*/\*\**

*\* @return the time stamp for when this location was recorded*

*\*/*

**public** String **getTimeStamp(){**

**}**

*/\*\**

*\* @return the original timestamp for the location*

*\*/*

**public** Calendar **getOldTime(){**

**}**

*/\*\**

*\* Converts the data stored in this object to a JSON string.*

*\**

*\* @return the JSON String.*

*\*/*

**public** String **toJSON(){**

}

**}**

## WTCKeyLocation class

*/\**

*\* @(#) WTCKeyLocation.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **WTCKeyLocation** **extends** WTCLocation **{**

*/\*\**

*\* Constructs a blank key location.*

*\*/*

**public** **WTCKeyLocation(){**

**}**

*/\*\**

*\* Constructs a key location with a given longitude and latitude.*

*\**

*\* @param longitude the longitude of the location.*

*\* @param latitude the latitude of the location.*

*\*/*

**public** **WTCKeyLocation(double** longitude**,** **double** latitude**){**

**}**

*/\*\**

*\* Adds the file path of a photo to this key location.*

*\**

*\* @param path the file path.*

*\*/*

**public** **void** **addPhoto(**String path**){**

**}**

*/\*\**

*\* Removes the file path of a photo from this hey location.*

*\**

*\* @param path the file path.*

*\*/*

**public** **void** **removePhoto(**String path**){**

**}**

*/\*\**

*\* @return the list of photo paths in this location*

*\*/*

**public** List**<**String**>** **getPhotos(){**

**}**

*/\*\**

*\* @return the name of this location*

*\*/*

**public** String **getLocName()** **{**

**}**

*/\*\**

*\* Sets the name of this location*

*\**

*\* @param locName the name of the location being set*

*\*/*

**public** **void** **setLocName(**String locName**)** **{**

**}**

*/\*\**

*\* @return the description of this location*

*\*/*

**public** String **getLocDesc()** **{**

**}**

*/\*\**

*\* Sets the description of this location*

*\**

*\* @param locDesc the description of the location being set*

*\*/*

**public** **void** **setLocDesc(**String locDesc**)** **{**

**}**

@Override

**public** String **toJSON(){**

**}**

**}**

## SendData class

*/\**

*\* @(#) SendData.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **SendData** **extends** AsyncTask**<**String**,** Void**,** Void**>** **{**

*/\*\**

*\* Constructs a SendData object.*

*\**

*\* @param activity a link back to the activity that made this object.*

*\* @param tour the tour being saved to the server*

*\*/*

**public** **SendData(**Activity activity**,** WTCTour tour**){**

**}**

*/\*\**

*\* Sends the tour data to the specified server.*

*\* <p>The tour data and the accompanying photos are placed into a*

*\* Multi-Part MIME and sent to the server using HTTP POST.</p>*

*\**

*\* @param params the list of URLs to send the tour to.*

*\*/*

@Override

**protected** Void **doInBackground(**String**...** params**)** **{**

**}**

*/\*\**

*\* Notifies the user that the tour was saved and takes them back to the home screen.*

*\*/*

@Override

**protected** **void** **onPostExecute(**Void result**)** **{**

**}**

**}**

## NewWalkFragment class

*/\**

*\* @(#) NewWalkFragment.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **NewWalkFragment** **extends** DialogFragment **implements** OnClickListener**{**

*/\*\**

*\* Creates the dialog box that collects the details for the walk.*

*\*/*

@Override

**public** Dialog **onCreateDialog(**Bundle savedInstanceState**)** **{**

**}**

*/\*\**

*\* Dictates what happens when the user clicks the positive or negative button on the*

*\* dialog.*

*\*/*

@Override

**public** **void** **onClick(**DialogInterface dialog**,** **int** which**)** **{**

**}**

**}**

## LocationsDetailsFragment class

*/\**

*\* @(#) LocationDetailsFragment.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **LocationDetailsFragment** **extends** DialogFragment **implements** OnClickListener**{**

*/\*\**

*\* Links this dialog back to the activity that created it.*

*\*/*

@Override

**public** **void** **onAttach(**Activity activity**){**

**}**

*/\*\**

*\* Creates the dialog and sets its layout.*

*\*/*

@Override

**public** Dialog **onCreateDialog(**Bundle savedInstanceState**)** **{**

**}**

*/\*\**

*\* Dictates what happens when you click a button on the dialog.*

*\*/*

@Override

**public** **void** **onClick(**DialogInterface dialog**,** **int** which**)** **{**

**}**

*/\*\**

*\* @return the text field for the location name.*

*\*/*

**public** EditText **getTfLocName()** **{**

**}**

*/\*\**

*\* @return the text field for the location description.*

*\*/*

**public** EditText **getTfLocDesc()** **{**

**}**

**}**

## NoNetworkFragment class

*/\**

*\* @(#) NoNetworkFragment.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **NoNetworkFragment** **extends** DialogFragment **implements** OnClickListener **{**

*/\*\**

*\* Create this dialog and sets its layout.*

*\*/*

@Override

**public** Dialog **onCreateDialog(**Bundle savedInstanceState**)** **{**

**}**

*/\*\**

*\* Does nothing in this instance, just needs to be here.*

*\*/*

@Override

**public** **void** **onClick(**DialogInterface dialog**,** **int** which**)** **{**

**}**

**}**

## FinishWalkFragment class

*/\**

*\* @(#) FinishWalkFragment.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **FinishWalkFragment** **extends** EndWalkFragment **{**

*/\*\**

*\* Creates the dialog and sets its layout.*

*\*/*

@Override

**public** Dialog **onCreateDialog(**Bundle savedInstanceState**)** **{**

**}**

*/\*\**

*\* Dictates what happens when you click a button on the dialog.*

*\*/*

@Override

**public** **void** **onClick(**DialogInterface dialog**,** **int** which**)** **{**

**}**

**}**

## EndWalkFragment class

*/\**

*\* @(#) EndWalkFragment.java Version 1.0*

*\* Copyright(c) Group 5 @Aberystwyth University Computer Science Dept: Yr 2 (2014)*

*\* All Rights Reserved*

*\*/*

**public** **class** **EndWalkFragment** **extends** DialogFragment **implements** OnClickListener**{**

*/\*\**

*\* Link the this dialog back to the activity creating it.*

*\*/*

@Override

**public** **void** **onAttach(**Activity activity**){**

**}**

*/\*\**

*\* Creates the dialog and sets its layout.*

*\*/*

@Override

**public** Dialog **onCreateDialog(**Bundle savedInstanceState**)** **{**

**}**

*/\*\**

*\* Dictates what happens when you click one the dialog buttons.*

*\*/*

@Override

**public** **void** **onClick(**DialogInterface dialog**,** **int** which**)** **{**

**}**

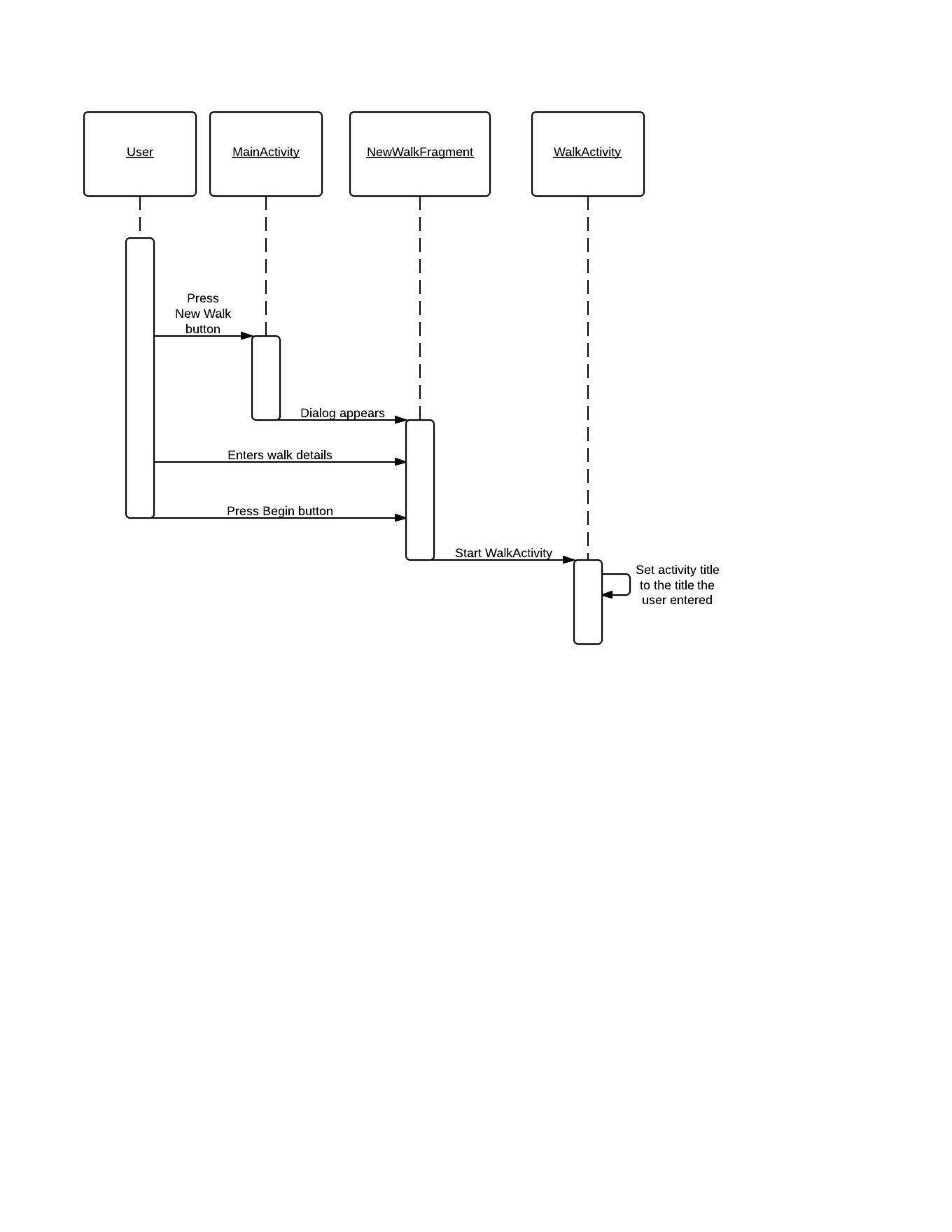
**}**

# Detail Design

This section contains information about the more difficult and not self-evident parts of the system design. It includes the sequence diagrams for the different parts of operation to describe in more detail the order in which events in the program are executed. Further explanation on the algorithms used by both the Android and the Server side of the system are explained to clarify how the system’s main components interact.

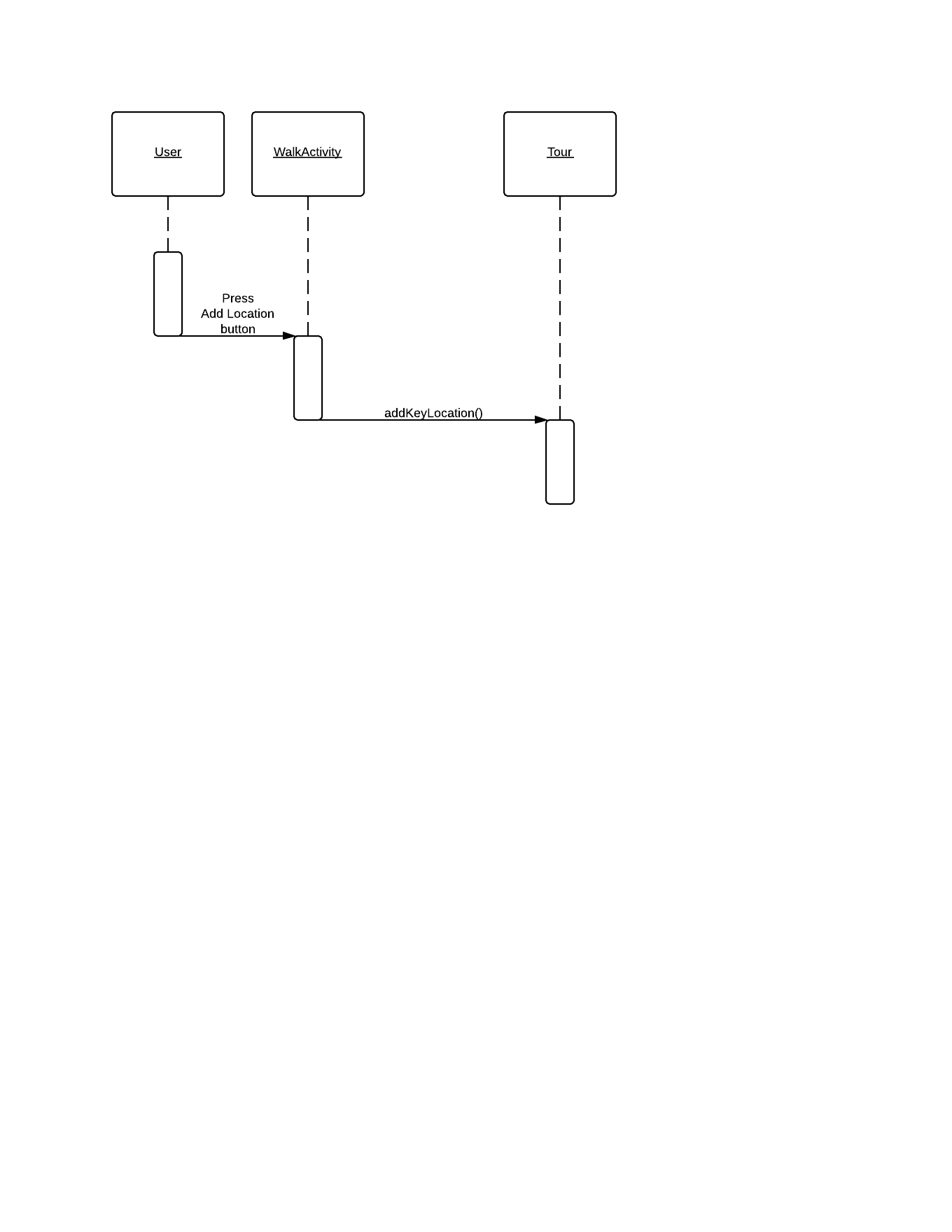
## Sequence Diagrams

This section contains the sequence diagrams for the whole system. They have been split into different segments representing different aspects of system operation.



### Tour creation sequence diagram

### Add key location sequence diagram



### Data transfer sequence diagram

### Server side sequence diagram

## Algorithm Description

### Server Side Data Delivery Algorithm

* Android app uploads MIME message to a PHP server page over the internet using the HTTP POST method;
* 2. The PHP file unpacks the image attachments and saves them to a temporary directory with a unique name (e.g. tmp/05012013my\_tour). It records success/ failure to the server log and if successful, will progress;
* 3. The PHP file extracts the data from the JSON file, checks whether the data is valid and continues;
* 4. It then produces an SQL query to insert the data into the database. It records success/ failure to the server log and if successful, will progress;
* The files are moved from the tmp directory to a permanent directory with a unique name. (The primary key of the tour in the database).

### Android side algorithms

* **Generating JSON:**

All relevant classes will have a toJSON() method that constructs a JSON object based on the contents of the class (Appendix B). This includes any objects constructed by the class itself. Any photographs that are going to use have their file path added as a component of the generated JSON objects.

* **Communicating with the server:**

The created JSON objects are sent as part of a MIME message via HTTP Post.

In Android the sending of information via HTTP Post is rather simple, we create an object known as a HTTP Post object, with a URL attached to it, we then add all associated information necessary and send it to the URL previously attached.

* **Keeping track of locations:**

To keep track of the route the user is taking we are going to generate a location every few meters determined by a persistent algorithm checking the difference in the longitude/latitude and calculating if the difference equates to or is greater than the prescribed difference for adding a location. If the user adds a location within 5 minutes of the app generating one the app generated one will be removed to reduce on data transfer for the user.

* **Location management:**

As stated above the previous location will be checked on the creation of a key location by the user if the user-generated location’s timestamp more than 5 minutes older than the previous location, just add the location to the end of the linked list; however if the previous non-user created location is less than 5 minutes old, replace it with the new user-generated location.

* **Photo management:**

The android operating system allows us to simply store our own images that the user will create in app in our own file storage system under the images folder on the device. This will allow us to easily attach the images for the relevant key locations while being able to just reference a file path in the JSON string.

# Data Structures

### Android data structures

The data structure for the application is essentially one class containing a linked list of objects of another class. More specifically it has a ‘Tour’ class which holds the linked list of ‘Location’ objects. The Tour class contains fields for the tour’s name, short description and long description; along with a field which is the linked list. The Location class holds fields for the latitude and longitude of the location; it also has a timestamp for when that location was recorded. There is also a ‘KeyLocation’ class which contains further information. More specifically it contains an array of file paths to the photos the user takes for that location, a name for that location and a short description for that location.

### Server side data structures

The tables in the database that will hold all the data for the walk:

* *List of Walks Table:*

|  |  |  |
| --- | --- | --- |
| Field Name | Field Description | Field Data Format |
| **id** | Primary Key (auto increment) | integer |
| **title** | Title of the tour | text |
| **shortDesc** | A short description of the tour (<100 characters) | text |
| **longDesc** | A detailed description of the tour. (<1000 characters) | text |
| **hours** | The number of hours the walk will take | float |
| **distance** | The total distance of the tour in kilometres | float |

* *Location Table :*

|  |  |  |
| --- | --- | --- |
| Field Name | Field Description | Field Data Format |
| **id** | Primary Key (auto increment) | integer |
| **walkID** | Foreign key, referencing the id field of the tour that the location is associated with | integer |
| **latitude** | The latitude map reference for the location | float |
| **longitude** | A detailed description of the tour. | text |
| **timestamp** | The time in hours from the beginning of the tour | float |

* *Place description table:*

|  |  |  |
| --- | --- | --- |
| Field Name | Field Description | Field Data Format |
| **id** | Primary Key (auto increment) | integer |
| **locationID** | Foreign key, referencing the location that the point of interest is referencing | integer |
| **description** | The description of this point of interest (<500 characters) | text |

* *Photo Usage Table:*

|  |  |  |
| --- | --- | --- |
| Field Name | Field Description | Field Data Format |
| **id** | Primary Key (auto increment) | integer |
| **placeID** | Foreign key, referencing the point of interest that the image is attached to | integer |
| **photoName** | The name of the jpg file for the photo (without “.jpg” suffix) | text |

## Entity Relationship Diagram



# APPENDIX **A**

The mime message will contain a “From” field which will store the user’s name and email (From: User’s Name <user@usershost.com>) and the name of the tour in the “Subject” field (Subject: My Tour). Writing the tour name to the subject field will allow the server to record the process in the log, even if there is an error with the JSON code. It will include a MIME version declaration of version 1.0 (MIME-Version: 1.0) and a multipart content type declaration (Content-Type: multipart/mixed; boundary=”part”). The JSON code will be stored in the only text type part. All of the images will be stored as attachments in jpeg format.

**Sample MIME message:**

From: John Doe <example@example.com>

Subject: TOUR NAME

MIME-Version: 1.0

Content-Type: multipart/mixed;

boundary="part"

--part

Content-Type: text/plain

***JSON CODE GOES HERE***

--part

Content-Type: image/jpeg;

Content-Disposition: attachment;

filename="file1.jpg"

jgfc,jbjytf,nmvk-0987y6trfgi9876trdfvbhjytrfdc

--part—

# APPENDIX **B**

* *Fields for root of JSON data set:*

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Format |
| **title** | The title of the walk | A string of <30 characters |
| **shortDesc** | A short description of the tour to be displayed in lists of tours on the website. | A string of <100 characters |
| **longDesc** | A long description of the tour to be displayed alongside the map on the website. | A string of <1000 characters |
| **route** | A sequence of GPS locations that describe the route of the tour | A collection of objects representing GPS coordinates. (See List of walks Table) |
| **locations** | A set of locations of interest along the tour. | A collection of objects representing locations of interest. (See Location Table) |
| **time** | The number of seconds that elapsed during the recording of the tour. (Not including when paused) | Integer |
| **distance** | The distance of the route of the tour in meters. | Integer |

* *Fields for location/ route objects:*

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Format |
| **id** | A unique ID indicating the index of the location in the sequence | Integer |
| **longitude** | The longitude of the current GPS location on the route | Integer |
| **latitude** | The latitude of the current GPS location on the route | Integer |
| **time** | The number of seconds that elapsed from the be-ginning of the tour to this recorded location. (Not including when paused) | Integer |

* *Fields for POI object:*

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Format |
| **coord** | The ID of the GPS coordinate object that the location is attached to | integer |
| **description** | A short description of the current location. | A string of <500 characters |
| **media** | A set of URLs pointing to the images to be associated with the location | A string collection of variable length. |

# Appendix **C**

**Sample JSON file:**

{

"Title": "My Walk",

"shortDesc": "A walk from grans house to my house",

"longDesc": "This is a walk that I take from my house to my nans. I hope you enjoy it…",

"route": [

{

"id": 0,

"longitude": 345674,

"latitude": 583848,

"time": 0

},

{

"id": 1,

"longitude": 345684,

"latitude": 583848,

"time": 5

},

***//LOTS MORE HERE…***

],

"pointOfInts": [

{

"coord": 5,

"description": "This is where I live",

"media": [

"file1.jpg",

"file2.jpg"

]

},

{

"coord": 17,

"description": "This is about half way",

"media": []

},

{

"coord": 25,

"description": "This is where my gran lives",

"media": [

"file3.jpg"

]

}

],

"time": 45676,

"distance": 23454

}

# References

[1] Software Engineering Group 05. Project Plan. S. Raychev, B. O’Donovan, H. Clark, W. Arslett, W. Lea, N. Vicker and S. Clasby. 1.2 Release.

[2] Software Engineering Group 05. Test Specification. S. Raychev, B. O’Donovan, H. Clark, W. Arslett, W. Lea, N. Vicker and S. Clasby. 1.1 Release.

[3] Software Engineering Group Projects. General Document Standards. C. J. Price, N. W. Hardy and B. P. Tiddeman. SE.QA.03. . 1.6 Release.

[4] Software Engineering Group Projects. Design Specification Standards. . C. J. Price, N. W. Hardy and B. P. Tiddeman. SE.QA.05A. . 1.7 Release.

[5] Software Engineering Group Projects. Java Coding Standards. . C. J. Price, A. McManus. SE.QA.09. . 1.7 Release.

# Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | CFF No. | Date | Changes made to the document | Changed by |
| 1.0 | N/A | 02.12.13 | N/A – First release of the Design Specification | srr11 |
| 1.1 | N/A | 05.12.13 | Updated Diagrams & Tables; Added algorithms | srr11 |
| 1.2 | N/A | 06.12.13 | Finalised the design specification | srr11 |
| 1.3 | N/A | 10.02.14 | Changed FR Table and Class/Interface info | bmo |
| 2.0 | #1 | 13.02.14 | Updated with Bernie’s remarks. | bmo |