Software Engineering Group Projects –

Design Specification

Author:	Harry Buckley
	Mosopefoluwa David Adejumo
	Martin Zokov
	Jack Reeve
	Ryan Gouldsmith
	Mark Pitman
	Zack Lott
	Mark Smith
	Maciej Dobrzanski

Config Ref:	SE.DS.01
Date:	6st December 2013
Version:	2.4
Status:	Release

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to specify the technical design of both the Android and web applications. It will go into detail regarding functions. This will allow us to more easily designate tasks to team members when it comes to coding week. It will also show how these functions interact with each other and how the website, server and Android app interact through the use of sequence diagrams. The document is structured in a way that makes it easy to refer to when the programmer needs clarification on how to build a certain function. The document will also show how the database will be structured and what the field names will be.

1.2. Scope

This document will cover all aspects of the Android and web design and their implementation. It should be read by all members of the group and approved by the client. It will be used as a guide for the programmers to build from in coding week. The document will allow the team leader to assign a given function to a team member which they can then code.

1.3. Objective

The precise areas which this document will cover are:

- Provide a clear class diagram, covering all aspects of the Android app.
- Define, in detail, the interaction between all the programs in the system.
- Provide a structure for implementation of the applications.
- Outline the significant systems to be used in the applications.
- Provide descriptions of functions.

2. Decomposition Description

2.1. Programs in System

The walk tour application consists of

- The Android application
- The Data Layer
- The website application

2.1.1. The Android Application

The Android application is used to create physical data representing a route allowing the users to record and upload a walk. It allows the user to add points of interest along a route and associate points of images. It displays a map screen and is used to record location data for a walk using GPS or AGPS (Assisted GPS). It also gives the user options to add pictures to a walk. When a walk is finished it is saved into a local SQL database using SQLite which is then sent to the server. [1]
Requirements Covered: (FR1, FR2, FR3, FR4, FR5, FR6, FR7, FR9, EIR1, PR1)

2.1.2. The Data Layer

Stores walk info in MySQL which it receives from the Android application as a MIME type. When the server application receives information for a walk it appends the location data to the database and stores all pictures on the server machine. The database server will also have a PHP file which handles the uploading of data from the Android device. The file that handles the upload can be accessed via the URL in a browser, but doing so will present an error message. [1]
Requirements Covered: (DC3)

- List of Walks relation:
  - id
  - title
  - shortDesc
  - longDesc
  - hours
  - distance
- Location
  - id
  - walkID
  - latitude
  - longitude
  - timestamp
- Place description
  - id
  - locationId
  - description
- Photo Usage
  - id
  - placeId
  - photoName

2.1.3. The Web Application

This allows the user to view walks. The website is also hosted on the data server and can be used for viewing information about walks including route taken, points of interest and pictures. This program overlaps with 1.1.2 (Database Server). It interacts with the database using PHP.
Requirements covered: (FR8, FR9) [1]

2.2. The Android Application

This section covers a list of the most important classes for each program. However, not all the classes used in the program are listed here. The complete set of classes can be seen in the class diagram – Section 4.1.2. These classes will all be written in Java.

2.2.1. WalkModel

A WalkModel holds all the data for a single walk, includes a list of all location points that trace the path and a list of all the places of interest.

2.2.2. RouteRecorder

The RouteRecorder retrieves the current location from the system, and depending on factors such as speed and direction, the location information will be added to the local WalkModel. This class will carry out some analysis of the path travelled so far to determine when to record points, i.e. if a recorded path seems to be travelling in a straight line then fewer point will be need added than if the path traces a circle.

2.2.3. WalkManager

WalkManager interacts with the local SQLite database, it used to both save and retrieve WalkModels from the local storage. Instances of this class are created as and when they are needed, it doesn't need to be passed between other objects.

2.2.4. FileTransferManager

A connection will be made with the server via the FileTransferManager. It is responsible uploading and downloading WalkModels, including all associated images, from the database server. This class only interacts with the WalkManager, so any objects wishing to upload of download content must connect through WalkManager, this is to add an extra layer of abstraction that simplifies the solution.

2.2.5. GeneralActivity

GeneralActivity is an abstract class that extends Activity. It defines the general layout of all the screens (MainMenuSreen, MapScreen, etc.). It provides subclasses with access to several static variables that describe the layout that allow changes such things as the background, and text colour. All the screens displayed to the user are subclasses of GeneralActivity.

2.3. Data Layer

The files here are used to control the interaction between the database and the other programs in the module. All these files will be written in PHP. Object Oriented Programming will not be implemented in this system.

2.4. Web Application

The following are files in PHP that will be used to interact between the database and the website. These are also pages that will be visible and accessible by the user unless otherwise stated. Object Oriented Programming will not be implemented in this system.

2.4.1. Index

This file will serve as the homepage and will display images selected at random from the database, offering a slideshow. If there are no images, a set of pre-set generic images will be used.

2.4.2. Walk_List

This file will process information from our database and display it as a list of walks. The walks will be clickable in order to view them in more detail. When listing information, a paging system will be used. I.e. each page will only display a limited number of walks.

2.4.3. Walk_Details

This file will be used to give the user a more in depth look at a specific walk.
This means they will be able to see a map view, images taken on the walk, and points of interest.
In the event of there being no long description, the short description is used, otherwise no short description is displayed.

2.4.4. Maps_Display

The Google Maps API will be used to portray a person's walk data into a visual map. The user will also be able to view points of interest on the map. This will serve as a separate file that will interact with Google's system.

2.4.5. File_Saver

The file saver uses the Apache HTTP Client to decode the data sent from the android application. This will mean our application will be able to 'POST' data to the server. This reduces load on the server compared to our previous idea of zipping and unzipping each set of files for a walk.

2.5. *Table Mapping Requirements Onto Classes*

This section gives an overview of what classes/files cover what requirements as specified by the client. [1]

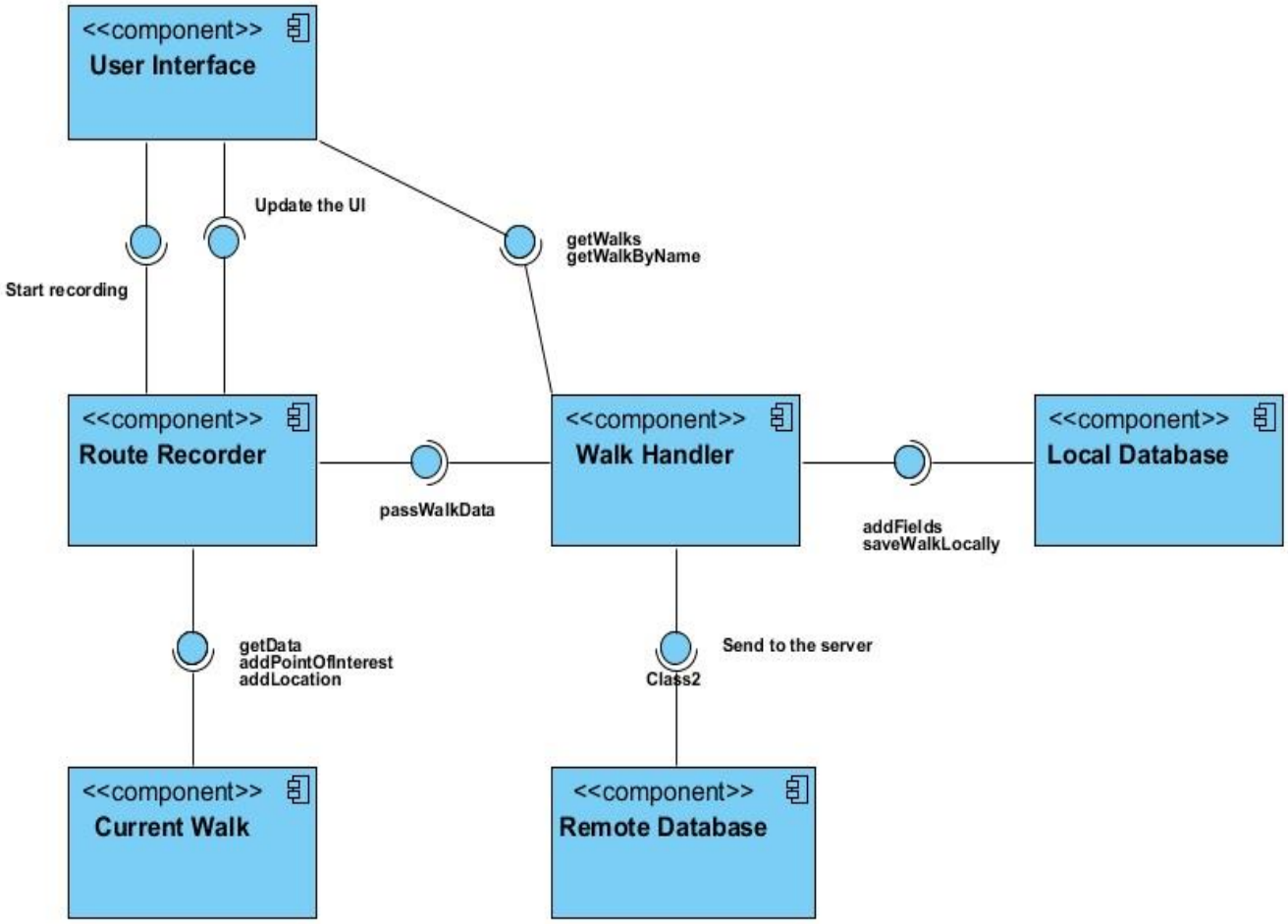
FR1	GeneralActivity, MapScreen, WalkSetupScreen, MyWalksScreen, MainMenuScreen, OptionsScreen, WalkInfoView
FR2	MapScreen, RouteRecorder, WalkModel
FR3	LocationPoint, PointOfInterest
FR4	LocationPoint, PointOfInterest
FR5	WalkInfoView
FR6	WalkManager, FileTransferManager
FR7	RouteRecorder
FR8	walk_details, walk_list
FR9	file_saver

3. Dependency Description

This section outlines how the modules and programs will interact and how they are linked together.

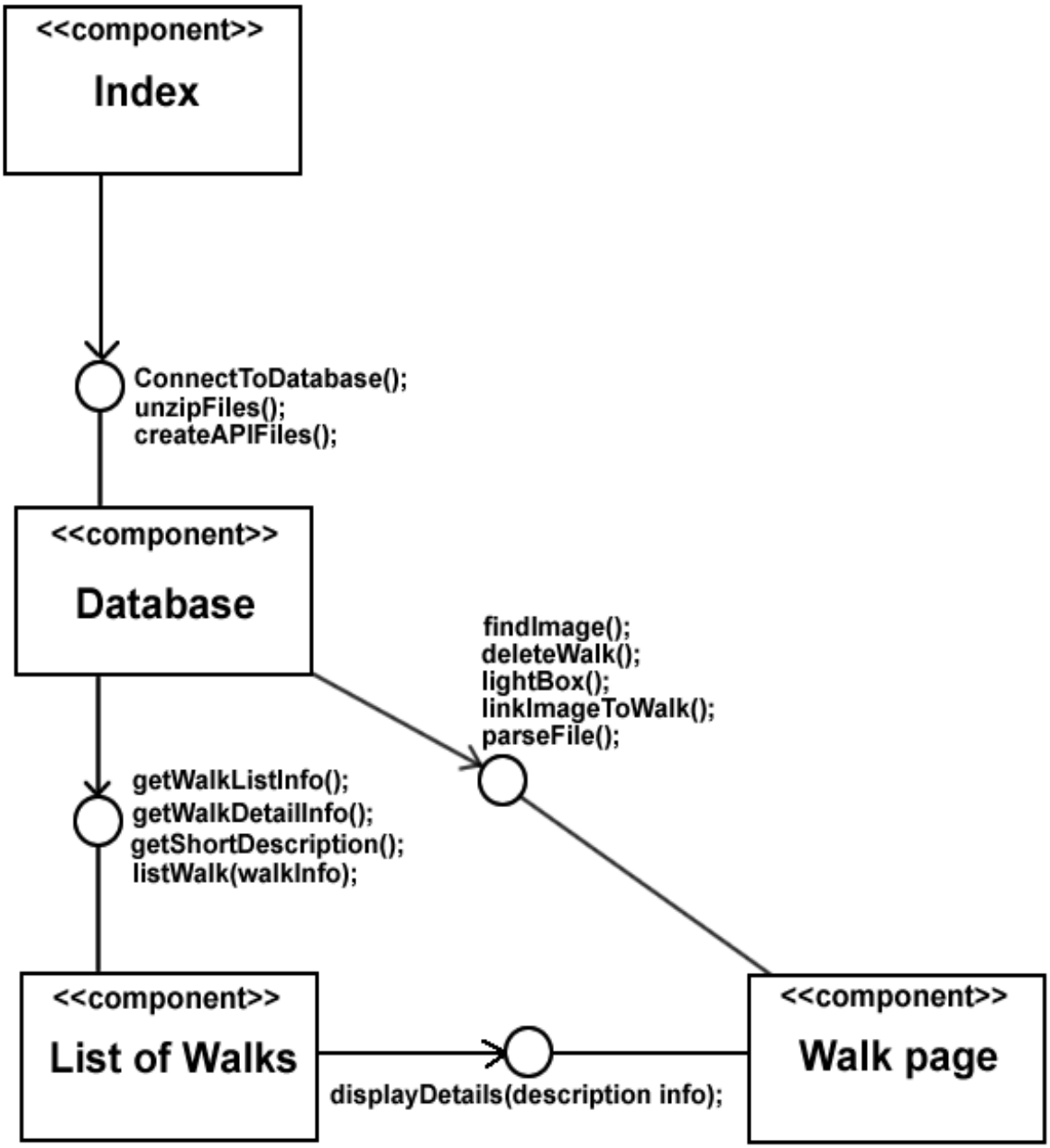
3.1. Component Diagrams

3.1.1. Android



(Fig 2.1)

3.1.2. Web Application



(Fig 2.2)

3.2. Inheritance Relationships

Walk Handler depends on Local Database and Remote Database

Route Recorder depends on Current Walk and Walk Handler

User Interface depends on Walk Handler and Route Recorder

Walk Page depends on List of Walks and Database

List of Walks depends on Database

3.3. Interaction Description

3.3.1. The Android Application

The User Interface allows the user to interact with the application. Once the user starts a walk, the Route Recorder handles all the actions performed during the on-going walk. It passes all the information to the Current Walk in real time. The Current Walk is used for resuming a walk if it is paused. A completed walk is handed to the Walk Handler which controls the storage and retrieval of all walks performed on the device. Walk Handler will also control how walks are stored and sent to the server. The Walk Handler will also handle the conversion of the walk into the data that will be sent to the server. The Local Database controls the storage of walks locally, for later uploading. The Remote Database refers to the Data Layer module (2.1.2)

3.3.2. The Web Application

The walk page list of walks and index all require information stored in the database. The index page will select random images from the database and store them in a predefined URL which the index page will display. The index page provides links to the list of walks. The List of Walks will gather all the stored walks from the database and list small amounts of information about the walk. When a walk is selected, the Walk Page will find the associated walk in the database and gather all the information about the walk which will be displayed to the user.

3.3.3. System Interaction Overview

The programs interact using the Data Layer. The Android application uses the FileTransferManager class to encode the data via Apache HTTP Client and send it to the server via a URL linking to the file_saver file. Once the file_saver receives a valid file, it decodes the data and stores it as required in the database. The walk_list, walk_detail and index files all request this information from the database as required.

4. INTERFACE DESCRIPTION

This section contains an implementation of different classes and files in the program. However, there is a possibility that the implementation in the final product may differ from what is displayed here.

4.1. Screens

The following classes all extend Activity and are used to control the display.

4.1.1. OptionsScreen

```
public class OptionsScreen extends GeneralActivity {
        /**
         * This method changes the background colour,
         * of all generalActivity subclasses.
         *       The parameter v, is the object that called the method.
         */
        public void changeBackgroundColor(View v);
}
```

4.1.2. MainMenuScreen

```
public class MainMenu extends GeneralActivity {

        /**
         * Starts a new MyWalkScreen activity,
         * and displays it to the user.
         *The parameter v, is the object that called the method.
         */
        public void startMyWalksScreen(View v);

        /**
         * Starts a new WalkSetupScreen activity,
         * and displays it to the user.
         * The parameter v, is the object that called the method.
         */
        public void startWalkSetupScreen(View v);

        /**
         * Starts a new OptionsScreen activity,
         * and displays it to the user.
         * The parameter v, is the object that called the method.
         */
        public void startOptionsScreen(View v);

        /**
         * Starts a new LoginScreen activity,
         * and displays it to the user.
         * The parameter v, is the object that called the method.
         */
        public void StartLoginScreen(View v);
}
```

4.1.3. MyWalkScreen

```
public class MyWalksScreen extends GeneralActivity {

        /**
         * open a WalkInfoView popup on selected walk
         * The parameter v, is the object that called the method.
         */
        public void viewWalk(View v);
}
```

4.1.4. WalkSetupScreen

```
public class WalkSetupScreen extends GeneralActivity {

        /**
         * Starts a new MapScreen activity, and displays it to the user.
         * The detail that the user has input,
         *  are passed to the new activity.
         * The parameter v, is the object that called the method.
         */
        public void startWalk(View v);
}
```

4.1.5. MapScreen

```
public class MapScreen extends GeneralActivity {

        /**
         * creates and displays a AddPoiView.
         * The parameter v, is the object that called the method.
         */
        public void addPOI(View v);

        /**
         * creates and displays a WalkFinishedView.
         * The parameter v, is the object that called the method.
         */
        public void finishWalk(View v);

        /**
         * creates and displays a PlacesVisitedView.
         * The parameter v, is the object that called the method.
         */
        public void showPlacesVisited(View v);

}
```

4.1.6. GeneralActivity

```
public abstract class GeneralActivity extends Activity{

        /**
         * changes the background color of all GeneralActivity
         * subclasses to the passed
         * value. The int c, represents a color.
         */
        public void setBackgroundColor(int c);
```

```
        /**
         * changes the foreground color of all
         * GeneralActivity subclasses to the passed
         * value. The int c, represents a color.
        */
        public void setForegroundColor(int c);


        /**
         * changes the text color of all GeneralActivity
         * subclasses to the passed value.
         * The int c, represents a color.
         */
        public void setTextColor(int c);
}
```

4.2. Views

4.2.1. MapView

```
public class WalkView extends MapFragment/*from google API*/{
        /**
         * sets the walks that is to be displayed
         */
        public void setWalk(WalkModel walk);

        /**
         * this method will cause the map 'window'
         *          to redraw the route on to itself.
         */
        public void updateWalk();
}
```

4.2.2. WalkInfoView

```
public class WalkInfoView extends PopupView{

        /**
         * creates a WalkInfoView instance.
         * The WalkModel that is passed to it
         *is displayed in in the popup.
         */
        public class WalkInfoView(WalkModel walk);
}
```

4.2.3. PopupView

```
public abstract class PopupView extends DialogFragment {
        /**
         * closes the popup view.
```

```
         * The parameter v, is the object that called the method.
         */
        public void closePopup(View v);
}
```

4.2.4. PoiInfoView

```
public class PoiInfoView extends PopupView{

        /**
         * creates a PoiInfoView instance. The PointOfInterest
         * that is passed to it
         *is displayed in in the popup.
         */
        public void PoiInfoView(PointOfInterest point);
}
```

4.2.5. PlacesVisitedView

```
public class PlacesVisitedView extends PopupView{

        /**
         * creates a PlacesVisitedView instance.
         * All the PointOfInterest from the
         *passed walk are displayed in a table.
         */
        public void PlacesVisitedView(WalkModel walk);

        /**
         * opens a PoiInfoView.
         * The parameter v, is the object that called the method.
         */
        public void getPoiInfo(View v);
}
```

4.2.6. WalkFinishedView

```
public class WalkFinishedView extends PopupView{

        /**
         * displays a screen displaying
         * a summary of the finished walk, and
         * shows various options to the user regarding the WalkModel.        */
        public void WalkFinishedView(WalkModel walk);

        /**
         * open a PoiInfoView
         * The parameter v, is the object that called the method.
         */
        public void uploadWalk(View v);
}
```

4.2.7. AddPoiView

```
public class AddPoiView extends PopupView{
        /**
         * displays an place description input popup,
         *  and gives it a link to the RouteRecorder
         */
```

```
        public void AddPoiView(RouteRecorder recorder);

        /**
         * creates a PointOfInterst out of the given
         * data (from text fields) and      add the point the the WalkModel
         * The parameter v, is the object that called the method.
         */
        public void submit(View v);

        /**
         * uses ImageHandler to open the photoLibrary,
         * the selected photo is then added to the PointOfInterest.
         * The parameter v, is the object that called the method.
        */
        public void getPhotoFromLibrary(View v);

        /**
         * uses ImageHandler to open the camera app,
         * the taken photo is then added to the PointOfInterest.
         * The parameter v, is the object that called the method.
         */
        public void getPhotoFromCamera(View v);
}
```

4.3. Models

4.3.1. WalkModel

public class WalkModel {

```
        /**
         * creates a WalkModel, with LocationPoints already set, it is used
         *by the WalkManager when loading walk from database.
         */
        public WalkModel(String title,Vector<LocationPoint> path,String shortDesc,String longDesc);

        /**
         * returns a vector of all the LocationPoint in the walk.
         */
        public Vector<LocationPoint> getRoutePath();

        /**
         * returns the running total of km travelled.
        */
        public double getDistance();

        /**
         * returns the elapsed time since the walk was started.
         */
        public double getTimeTaken();

        /**
         * returns the name of the walk.
```

```
        */
        public String getTitle();

        /**
         * returns a short description of the walk
         */
        public String getShortDescription();

        /**
         * set the short description of the walk.
         */
        public void setShortDescription(String newShortDesc);

        /**
         * returns a long description of the walk.
         */
        public String getLongDescription();

        /**
         * set the long description for the walk.
         */
        public void setLongDescription(String newLongDesc);

        /**
         * adds a LocationPoint to the walk.
         */
        public void addLocation(LocationPoint point);
}
```

4.3.2. LocationPoint

public class LocationPoint {

```
        /**
         * creates a new LocationPoint
         */
        public LocationPoint(double x,double y);

        /**
         * creates a LocationPoint,
         *  used to recreate a point stored in the
         *database.
         */
        public LocationPoint(double x,double y,double time);

        /**
         * returns the time at which the point was recorded.
         */
        public double getTime();

        /**
         * returns the longitude, the east/west
         * distance from Greenwich.
         */
        public double getLongitude();

        /**
```

```
        * returns the latitude, the north/south distance from the equator.
        */
        public double getLatitude();

        /**
        * returns the distance between itself and a passed point.
        */
        protected double distanceTo(LocationPoint point);
}
```

4.3.3. PointOfInterest

```
public class PointOfInterest extends LocationPoint{

        /**
        * creates a PointOfInterest, at position x,y.
        *  The time is set automatically
        */
        public PointOfInterest(double x,double y);

        /**
        * creates a PointOfInterest, at position x,y.
        * The time is also explicitly defined, this is
        * used when creating a PointOfInterest from a database entry.
        */
        public PointOfInterest(double x,double y,double time);

        /**
        * returns all the images associated with this point.
        */
        public Vector<ImageInformation> getImages();

        /**
        * returns the description of this place.
        */
        public String getDescription();

        /**
        * sets the description of this point.
        */
        public void setDescription(String desc);
}
```

4.4. Controllers

4.4.1. RouteRecorder

```
public class RouteRecorder extends Service implements LocationListener{

        /**
        * creates a RouteRecorder instance,
        *  with the MapView that will display the walk.
        */
        public RouteRecorder(MapView map);
```

```
        /**
         * starts the recording of location points
         */
        public void startRecording();

        /**
         * adds a PointOfInterest to the recorded path.
         */
        public void savePoi(PointOfInterest poi);

        /**
         * stops the recoding of locations.
         */
        public void finish();
}
```

4.4.2. WalkManager

```
public class WalkManager extends SQLiteOpenHelper{

        /**
         * creates a new WalkManager.
         */
        public WalkManager(Context context);

        /**
         * adds the passed WalkModel to the local database.
         */
        public void addWalkModel(WalkModel walk);

        /**
         * returns the WalkModel with an id matching the passed one.
         */
        public WalkModel getWalkByID(int index);

        /**
         * uploads the given walk to the server,
         *  the server interaction is handled by the FileTransferManager.
         */
        public void uploadWalk(WalkModel walk);

        /**
         * returns the requested walk from the server,
         * the server interaction is handled by the FileTransferManager.
         */
        public WalkModel getWalkFromServerById();

}
```

4.4.3. FileTransferManager

```
public class FileTransferManager{

        /**
         * makes a connection to data server and
         *  uploads all files belonging to the given
         * file, the return values will be zero if
         *  the method succeeded without problems.
```

```
        */
        public int uploadWalk(WalkModel walk);
}
```

4.5. PHP Functions

4.5.1. General Functions

These functions will be included in multiple files

```
/**
*Appends data to the database
*@param data The data that will be passed into the database.
*@param field The field where the data is to be added
*/
appendToDatabase(data, field);

/**
*Finds the database and opens the connection
*/
connectToDatabase();

/**
*Closes the connection to the database
*/
closeConnection();

/**
*Removes a walk from the database
*@param walkID ID of the walk that will be removed
*/
deleteWalk(walkID);
```

4.5.2. file_saver.php

```
/**
*Links all images to the specified walk
*@param image The image that will be linked
*@param walk The walk to be linked to
*/
linkImageToWalk(image, walk);
```

4.5.3. walk_list.php

```
/**
*Gets all images associated with the walk
*@param walkID The ID of the walk to find images for
*/
findImage(walkID);

/**
```

***Gets the WalkID, Title, Location and Thumbnail Image of the walk*
*/**
getWalkListInfo();
/**
***Gets the short description of the walk*
*/**
getShortDescription();

4.5.4. walk_detail.php

/**
***Gets the Longs description, time taken, coordinates and images associated with a walk*
*/**
getWalkDetailInfo();
getShortDescription();
getWalkListInfo();

4.5.5. index.php

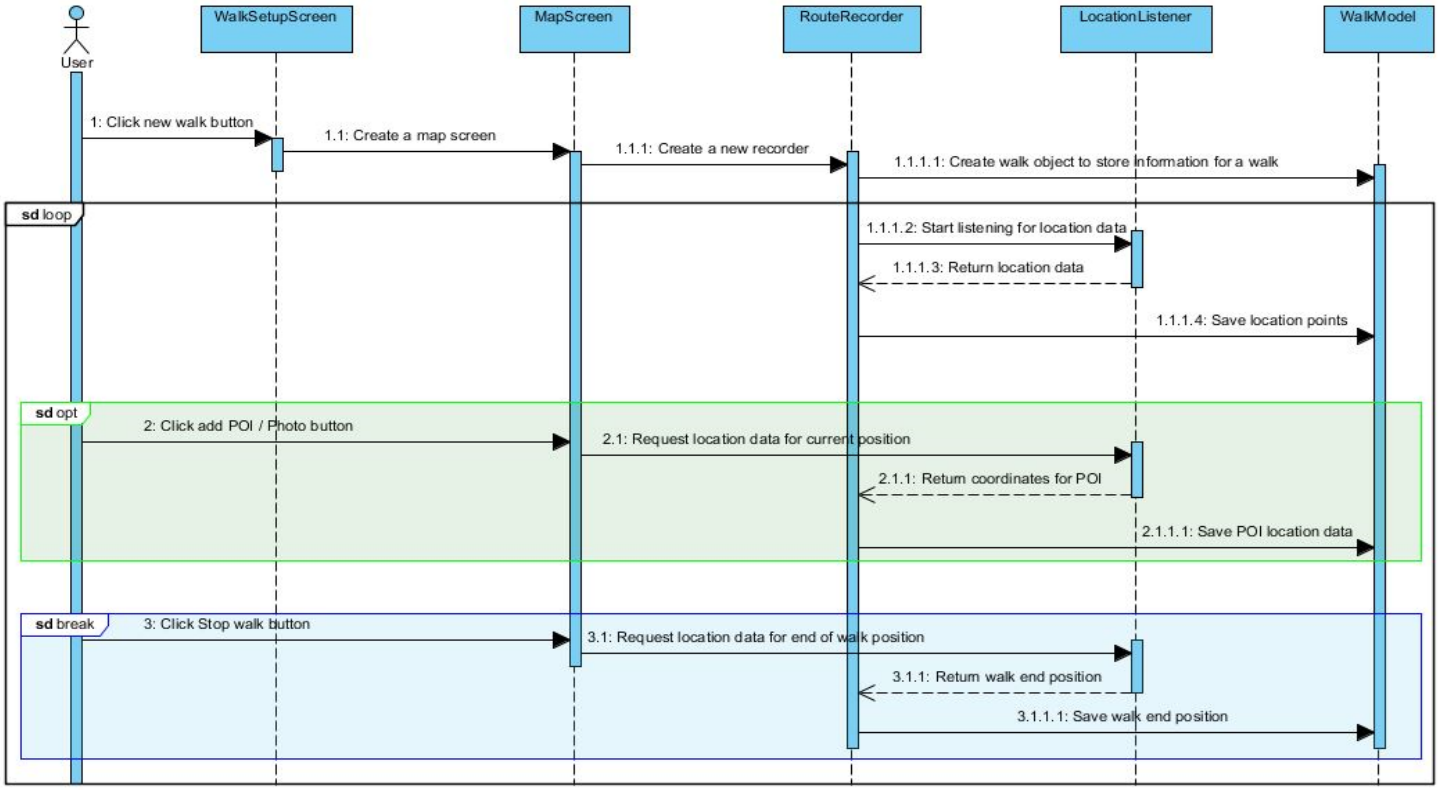
*/***
**Gets random images and assigns them to a URL*
/
findImage();

5. DETAILED DESIGN

This section details the algorithms and interactions that will be implemented in the program. The algorithms used may differ from the final product.

5.1. UML Diagrams

5.1.1. Sequence Diagram For Android



(Fig 4.1)

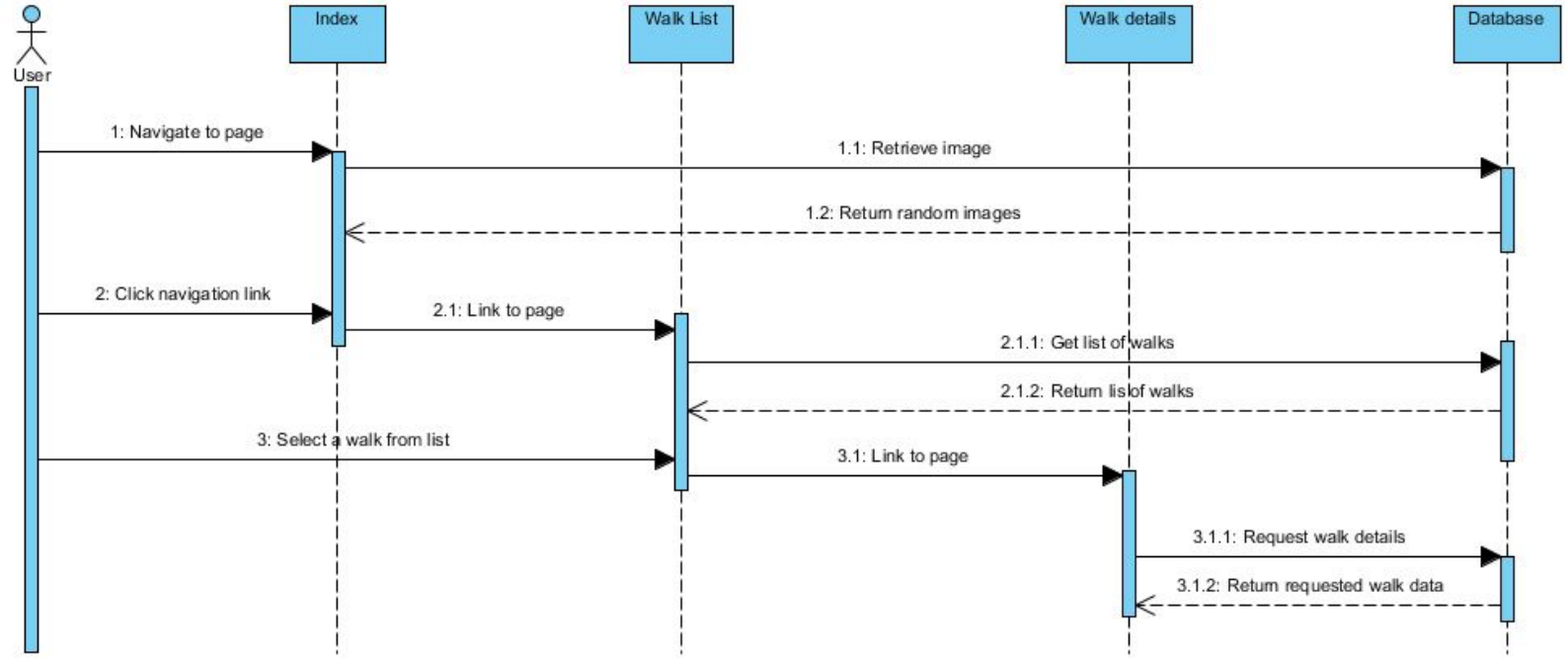
The sequence diagram describes the recording of a walk and how the classes which are involved in the process interact.[3]
In action 1. the user is prompted for details in the WalkSetupScreen and after he/she presses the start walk button, a map screen is shown and a RouteRecorder and WalkModel objects are created. After that the application goes into a loop of actions from the RouteRecorder, LocationListener and WalkModel classes. The recorder asks the listener for location data and when the data is returned, it is saved in the WalkModel's array of location points.

Action 2 is optional for the user, because it is not mandatory to have a Point of interest or photos in every walk. If a user decides to click the “Add POI” button, the LocationListener gives the coordinates of the current location to the RouteRecorder and they are saved in the WalkModel object.

Action 3 is the exit point of the loop for the current walk recording. It is done by clicking the stop button which breaks the loop and saves the last set of coordinates for the current walk.

The LocationListener is deliberately not activated at all times while a walk is in progress in order to save battery life.

5.1.2. Sequence Diagram For Web



(Fig 4.2)

The sequence diagram describes the user interaction with the website, and the websites interaction with the database.

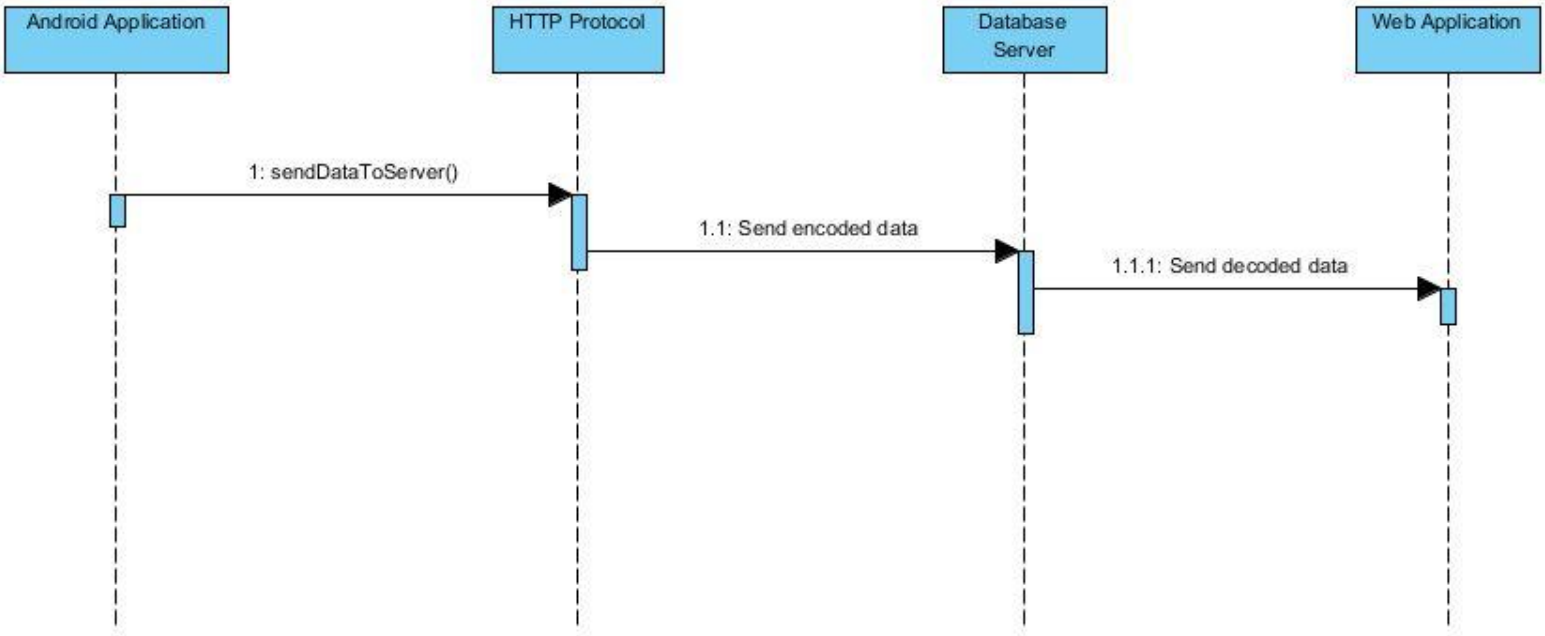
In action 1, the user navigates to the index page either via a link or via the URL. As soon as the user navigates to the homepage, the index file fetches images from the database to display to the user.

In action 2, the user navigates to a page where a list of walks is displayed. The list is paged to prevent extended scrolling in a situation where there are a large number of walks stored. The only information gathered from the database will be the walks location, title, short description and thumbnail image if possible.

In action 3 the user can view a selected walk. The file will fetch in addition to the data fetched in action 2, the long description, all images associated with the walk, the duration of the walk and all the points of interest.

Both the walk details and the walk list page can link back to the index page. All pages can easily be accessed via the URL however, if the user attempts to visit the walk details page via the URL, they will be redirected to the walk list page and given an error message.

5.1.3. Overall Interaction Sequence diagram



(Fig 4.3)

This diagram details the sequence in which data is sent and retrieved and how the Web Application and Android Applications interact with the database. Not all these interactions may be completed in one sitting. The Android application and Web Application interact independently of each other.

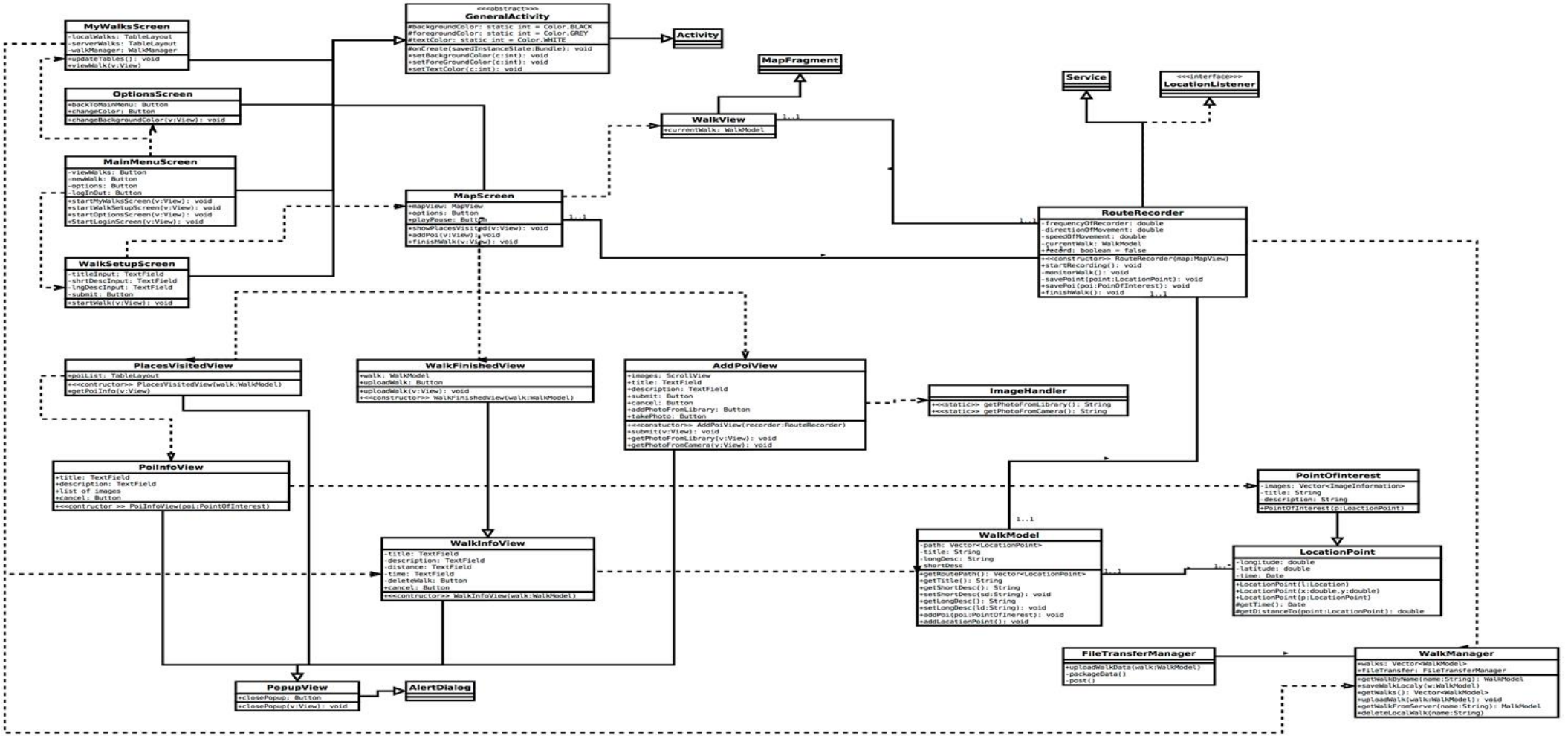
In action 1 the Android Application gathers all the information about a walk including images and sends it to the HTTP Protocol

In action 2 the HTTP Protocol, via the Apache HTTP Client, encodes the data for sending to the database.

In action 3 The Data Layer decodes the Android data and stores it in the database. On request the Data Layer retrieves the data from the database and sends it to the Web Application

In action 4 The Web Application requests information that is retrieved from the database. In the event that such information is not found, the Web Application displays an error to the user.

5.2. Class Diagram



The classes ending in ‘screen’, are all Activities. They all, in some way, display a layout to the screen and respond to user input. Any response that requires further processing would be passed to another class and then handed back to be displayed, but it would be the ’screen’ class itself that initialised the action.

There are several classes that have been suffixed with ‘View’, these classes all extend the android class View. They are all visible to the user and act much like ‘screen’ classes except that they don’t use the whole screen and do not change the displayed screen only create new Views.

The classes WalkModel,PointOfInterest and Location can all be considered to be model classes. There are used to store the walks data in an organised fashion, and have no methods to do anything other than to set and get information.

WalkManager, ImageHandler and FileTransferManager all perform some tasks that are not immediately apparent the user. They are the utility classes that are used by others.

5.3. Significant Algorithms

This section contains pseudocode and basic examples of algorithms that will be implemented in the final program. The final product may differ slightly from what is mentioned here.

5.3.1. Android Algorithms

5.3.1.1. RouteRecorder Algorithm

While(walk not finished***)*** *do*

get location

if(*distance between(new location, old location****) bigger than*** *X* ***) then***

add *new location to walkModel*

end if

5.3.2. PHP Algorithms

This section contains psuedocode of important functions for PHP files running on the server

5.3.2.1. Connect To The Database

```
/**
* This code will connect to our own database with our database name,
* username and password
*/
function connectToDatabase(){
        $con=mysqli_connect("db.dcs.aber.ac.uk",
"csgp07_13_14","csadmgp07","c54admgp07");
        /*
        * If the php fails to connect to the database this will appear
        */
        if (mysqli_connect_errno()){
                echo "Failed to connect to MySQL: " .
                        mysqli_connect_error();
        }
}
function closeDatabaseConnection(){
        mysqli_close($con);
}
```

5.3.2.2. 4.1.2.2. Unzip File

```
/**
*Code to Unzip data sent from android,
*however likely to change from zip folder to http
```

```
*/

function unzipFile(){
        $zip = new ZipArchive;

        if ($zip->open('test.zip') === TRUE) {
                $zip->extractTo('/my/destination/dir/');
                $zip->close();
                echo 'ok';
        }
        else{
                echo 'failed';
        }
}
```

5.3.2.3. Append To The Server Database

```
/**
* List all the fields that you wish to append to the database
*/
function connectToDatabase(){

$sql="INSERT INTO csgp07_13_14  (ShortDescription,
        LongDescription, PointOfInterest, Coordinates, Title)
                VALUES('$_POST[ShortDescription]',
                        '$_POST[LongDescription]','$_POST[PointOfIntere',
                                '$_POST[Coordinates]', '$_POST[Title]')";
        if (!mysqli_query($con,$sql)){
                die('Error: ' . mysqli_error($con));
        }
        echo "1 record added";
}


/**
* getWalkListInfo(); is used to get the
* information of the columns you wish to show
*/
```

5.3.2.4. Get Walk Info From The Database

```
function getWalkListInfo(){
$result = mysqli_query($con,"SELECT * FROM csgp07_13_14");

//Add Check button code
```

```
        echo "<table border='1'>
        <tr>
                <th>ShortDescription</th>
                <th>Location</th>
                <th>Title</th>
        </tr>";

        while($row = mysqli_fetch_array($result)){
                echo "<tr>";
                echo "<td>" . $row['ShortDescription'] . "</td>";
                echo "<td>" . $row['Location'] . "</td>";
                echo "<td>" . $row['Title'] . "</td>";
                echo "</tr>";
        }
        echo "</table>";
}


/**
* This will be the table that is shown when the user has selected a walk.
* It will display a new table showing new columns
*/
function getWalkDetailInfo(){

        $result = mysqli_query($con,"SELECT * FROM
                csgp07_13_14");

        echo "<table border='1'>
        <tr>
        <th>LongDescription</th>
        <th>Images</th>
        <th>Coordinates</th>
        <th>PointOfInterest</th>
        </tr>";

        while($row = mysqli_fetch_array($result)){
                echo "<tr>";
                echo "<td>" . $row['LongDescription'] . "</td>";
                echo "<td>" . $row['Images'] . "</td>";
                echo "<td>" . $row['Coordinates'] . "</td>";
                echo "<td>" . $row['PointOfInterest'] . "</td>";
                echo "</tr>";
        }
        echo "</table>";
}
```

```
/**
* Get the short description of the walk
*/
function getShortDescription(){

check to see if Long description != null
      if (long descrition == null){
            Display short description
        }
        else{
            return long description
        }

}
```

5.3.2.5. Delete Walk

```
/**
* This part of the code should be placed before the table
*/
function deleteWalk(){
        <form action="delete" method="post">
        /**
        *Code to delete the Information from the database
        */
        echo '<td><input type="submit" name="deleteItem" value="'.
              $row['id'].'" /></td>';
}
```

5.4. Significant Data Structures

5.4.1. WalkModel

This is the most significant data structure in the Android application. It contains the information for the route taken,
all of the GPS coordinates that the user has walked through, Points of interest.

5.4.2. LocationPoint

This class is responsible for storing a point on the map. It has variables for longitude, latitude and a timestamp.
After a GPS reading is taken for the current physical location is taken, it is put in an object of this class and stored in the WalkModel.

5.4.3. PointOfInterest

This data structure is used when adding a point of interest. It holds information for the description and title of a POI.
The class extends the LocationPoint so a POI can have location coordinates and a time stamp.

6. REFERENCES

[1] Software Engineering Group Projects. Requirements Specification. C. J. Price and B.P.Tiddeman. 1.2 (Release). 7 November 2013

[2] Software Engineering Group Projects. Design Specification Standards. C. J. Price and N. W. Hardy. SE.QA.05A. . 1.6. Release.

[3]Software Engineering Group Projects. Project Plan. Mosopefoluwa David Adejumo. 1.8 (Release). 6th November 2013

7. DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes Made to document*	*Changed by*
1.0	N/A	28/11/13	Created original document	HFB1
1.1	N/A	1/12/13	Added sections created by other members. Updated config reference Updated layout.	MDA
1.2	N/A	1/12/13	Fixed some formatting issues, added information to what fields will be used	RYG1
1.3	N/A	4/12/13	Added a new sequence diagram and a description for section 1.2	MVZ
1.4	N/A	5/12/13	Updated section 1.1. Added descriptions to all sections	MDA
1.5	N/A	5/12/13	Added a sequence diagram for the web.	MRP2
1.6	N/A	5/12/13	Updated class diagram, added FileTransferManager interface.	HFB1
1.7	N/A	6/12/13	Added Apache HTTP Client description.	JAR39
1.8	N/A	6/12/13	Added methods to the MapView interface.	HFB1
1.9	N/A	6/12/13	Changed sequence diagram for web and added overall interaction sequence diagram	MVZ
2.0	N/A	6/12/13	Updated the Introduction section.	JAR39, MRP2
2.1	N/A	6/12/13	Added web app diagram and Significant algorithms	ZAL
2.2	N/A	6/12/13	Updated author list. Updated formatting. Merged different versions of the document	MDA
2.3	N/A	6/12/13	Added image reference numbers. Updated images from MWD5 and MAS69. Added missing images. Added images and descriptions to section 3. Added references.	MDA
2.4	N/A	6/12/13	Formatting corrections. Changed version	MDA