
Group Project 07 Design Specification

Authors: Mosopefoluwa David Adejumo
Ryan Gouldsmith
Harry Flynn Buckley
Zack Lott
Mark Radcliffe Pitman
Jack Alexander Reeve
Mark Alexander Smith
Martin Vasilev Zokov
Maciej Wojciech Dobrzanski

Config ref: SE_07_PM_01

Date February 16, 2014

Version 2.7

Status Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Copyright ©
Aberystwyth University 2013

Contents

1	INTRODUCTION	4
1.1	Purpose	4
1.2	Scope	4
1.3	Objective	4
2	ARCHITECTURAL DESCRIPTION	5
2.1	Programs In System	5
2.1.1	The Android Application	5
2.1.2	The Database Server	5
2.1.3	Web Application	6
2.2	Significant Classes	6
2.2.1	Android Application	6
2.2.1.1	WalkModel	6
2.2.1.2	RouteRecorder	6
2.2.1.3	FileTransferManager	7
2.2.1.4	GeneralActivity	7
2.2.2	Database Server	7
2.2.3	Web Application	7
2.2.3.1	Index	7
2.2.3.2	Walk List	7
2.2.3.3	Walk Details	7
2.2.3.4	Google Maps Api	8
2.2.3.5	File_Saver	8
2.3	Table Mapping Requirements Onto Classes	8
3	DEPENDENCY DESCRIPTIONS	9
3.1	Component Diagrams	9
3.1.1	Android	9
3.1.2	Website	9
3.1.3	Database Server	10
4	INTERFACE DESCRIPTION	10
4.1	Screens	11
4.1.1	OptionsScreen	11
4.1.2	MainMenuScreen	11
4.1.3	MyWalkScreen	12
4.1.4	WalkSetupScreen	12
4.1.5	MapScreen	12
4.1.6	GeneralActivity	13

4.2	Views	14
4.2.1	MapView	14
4.2.2	WalkInfoView	14
4.2.3	PopupView	14
4.2.4	PoiInfoView	14
4.2.5	PlacesVisitedView	15
4.2.6	WalkFinishedView	15
4.2.7	AddPoiView	16
4.3	Models	16
4.3.1	WalkModel	16
4.3.2	LocationPoint	18
4.3.3	PointOfInterest	19
4.4	Controllers	20
4.4.1	RouteRecorder	20
4.4.2	FileTransferManager	21
5	DETAILED DESIGN	21
5.1	UML Diagrams	21
5.1.1	Android Sequence Diagram	21
5.1.2	Sequence Diagram For Web	23
5.1.3	Overall Interaction Sequence Diagram	23
5.2	Class Diagram	23
5.3	Significant Algorithms	25
5.3.1	Android Algorithms	25
5.3.1.1	RouteRecorder Algorithm	25
5.3.2	PHP Algorithms	25
5.3.2.1	Connect To The Database	25
5.3.2.2	Append To The Server Database	26
5.4	TODO CHECK THE ALGORITHMS	28
5.5	Significant Data Structures	28
5.5.1	WalkModel	28
5.5.2	LocationPoint	28
5.5.3	PointOfInterest	28
6	REFERENCES	29
7	DOCUMENT HISTORY	30

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 ARCHITECTURAL DESCRIPTION

2.1 Programs In System

The walk tour application consists of:

- The Android application.
- The Data server.
- 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. It also gives the user options to add pictures to a walk.

Requirements Covered: (FR1, FR2, FR3, FR4, FR5, FR6, FR7,FR9, EIR1, PR1)

2.1.2 The Database Server

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.

Requirements Covered: (DC3)

- List of Walks relation:
 - id
 - title
 - shortDesc
 - longDesc
 - hours
 - distance
- Location
 - id

- walkID
 - latitude
 - longitude
 - timestamp
- Place description
 - id
 - locationId
 - name
 - description
- Photo Usage
 - id
 - placeId
 - photoName

2.1.3 Web Application

Allows the user to view walks in more detail. 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)

2.2 Significant Classes

2.2.1 Android Application

This section describes the most significant classes in the application. The complete set of classes can be seen in the class diagram Section 4.1.2. These classes will all be written in Java.

2.2.1.1 WalkModel

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

2.2.1.2 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 traveled so far to determine when to record points,i.e. if a recorded path seems to be traveling in a straight line then fewer point will be need added than if the path traces a circle.

2.2.1.3 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 or download content must connect through WalkManager, this is to add an extra layer of abstraction that simplifies the solution.

2.2.1.4 GeneralActivity

GeneralActivity is an abstract class that extends Activity. It defines the general layout of all the screens (MainMenuScreen, MapScreen, etc.). It provides subclasses with access to several static variable 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.

2.2.2 Database Server

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.

2.2.3 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.

2.2.3.1 Index

This file will serve as the homepage and holds links to view the list of walks and terms of service.

2.2.3.2 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. Users will be able to select a walk via this file

2.2.3.3 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.

2.2.3.4 Google Maps Api

The Google Maps API will be used to portray a persons 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 Googles system.

2.2.3.5 File_Saver

The Apache HTTP Client will be used for by the android application to send data to our database server. 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. The data will be sent as a JSON string. This file will decode the JSON string and add all the walk data to the appropriate tables where required

2.3 Table Mapping Requirements Onto Classes

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

FR1	GeneralActivity, MapScreen, WalkSetupScreen, My-WalksScreen, MainMenuScreen, OptionsScreen, Walk-InfoView
FR2	MapScreen, RouteRecorder, WalkModel
FR3	LocationPoint, PointOfInterest
FR4	LocationPoint, PointOfInterest
FR5	WalkInfoView
FR6	WalkManager, FileTransferManager
FR7	RouteRecorder

3 DEPENDENCY DESCRIPTIONS

3.1 Component Diagrams

3.1.1 Android

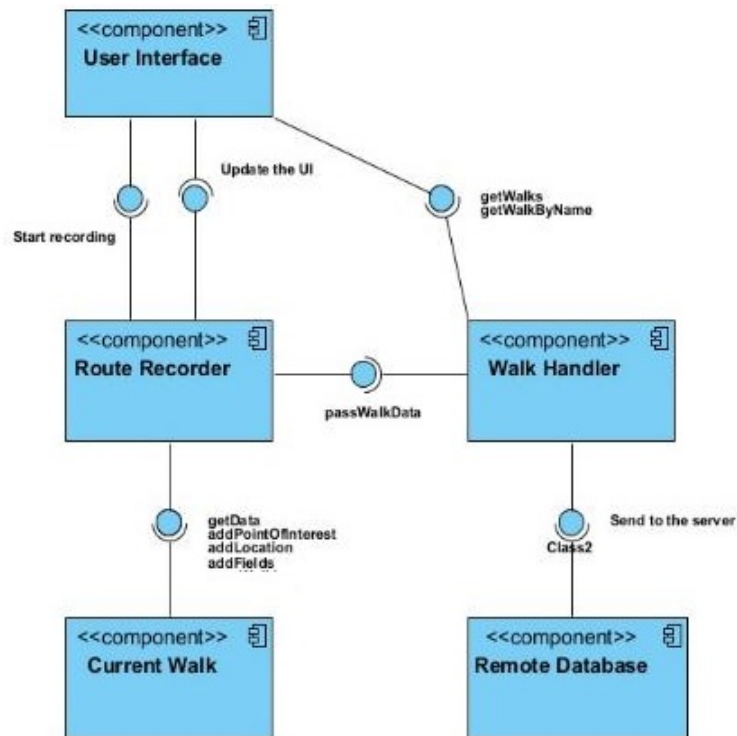


Figure 1: Android Dependency Diagram

3.1.2 Website

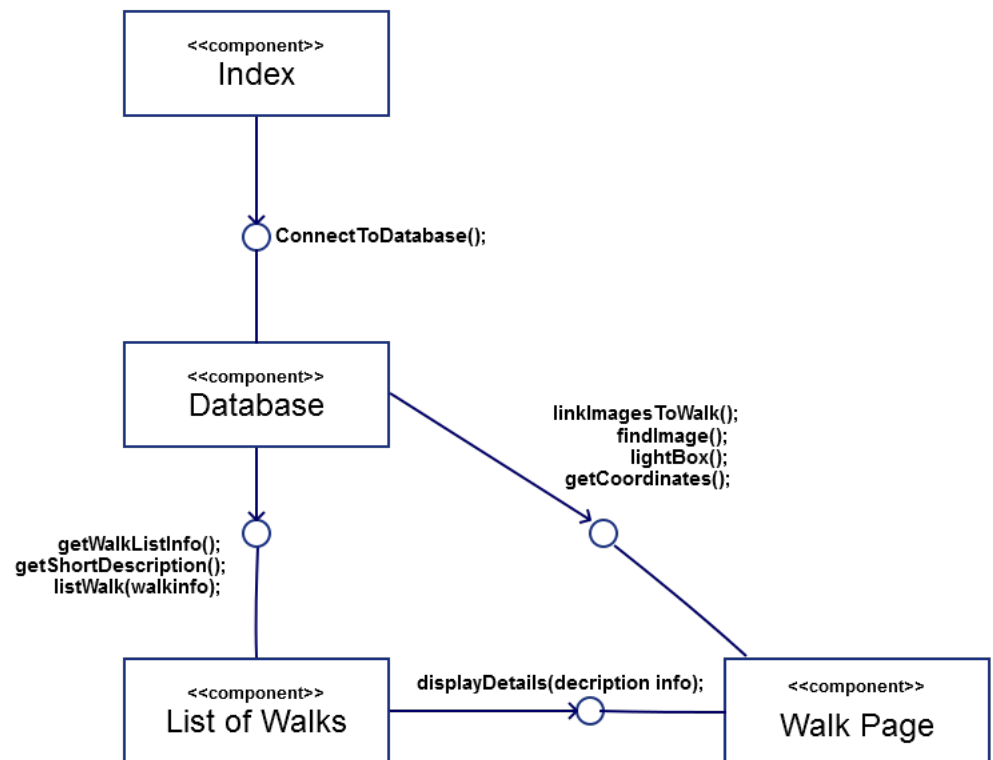


Figure 2: Web Component Diagram

3.1.3 Database Server

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 color ,  
     * of all generalActivity subclasses  
     * @param 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.  
     * @param v, is the object that called the method.  
     */  
    public void startMyWalksScreen(View v);  
  
    /**  
     * Starts a new WalkSetupScreen activity ,  
     * and displays it to the user.  
     * @param v, is the object that called the method.  
     */  
    public void startWalkSetupScreen(View v);  
  
    /**  
     * Starts a new OptionsScreen activity ,  
     * and displays it to the user.  
     * @param v, is the object that called the method.  
     */  
    public void startOptionsScreen(View v);  
  
    /**  
     * Starts a new LoginScreen activity ,
```

```
        * and displays it to the user.
        * @param 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
     * @param 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.
     * @param 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.
     * @param v, is the object that called the method.
     */
    public void addPOI(View v);

    /**
     * creates and displays a WalkFinishedView.

```

```
        * @param v, is the object that called the method.
        */
    public void finishWalk(View v);

    /**
     * creates and displays a PlacesVisitedView.
     * @param 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 MapView/* our class */ 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.  
     * @param 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.
 * @param 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
 * @param 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 PointOfInterest out of the given  
     * data (from text fields) and add the point the  
     * the WalkModel  
     * @param 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.  
     * @param 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.  
     * @param 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);

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

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

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

/**
 * @return the name of the walk.
 */
public String getTitle();

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

/**
 * set the short description of the walk.
 */
```

```
public void setShortDescription(String newShortDesc
    );

/**
 * @return 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)
        ;

    /**
     * @return the time at which the point was recorded
     * .
     */
    public double getTime();
}
```

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

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

/**
 * @return 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);

    /**
```

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

      /**
       * @return 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 FileTransferManager

```
public class FileTransferManager{  
  
    /**  
     * @param walk  
     * 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);  
}
```

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 Android Sequence Diagram

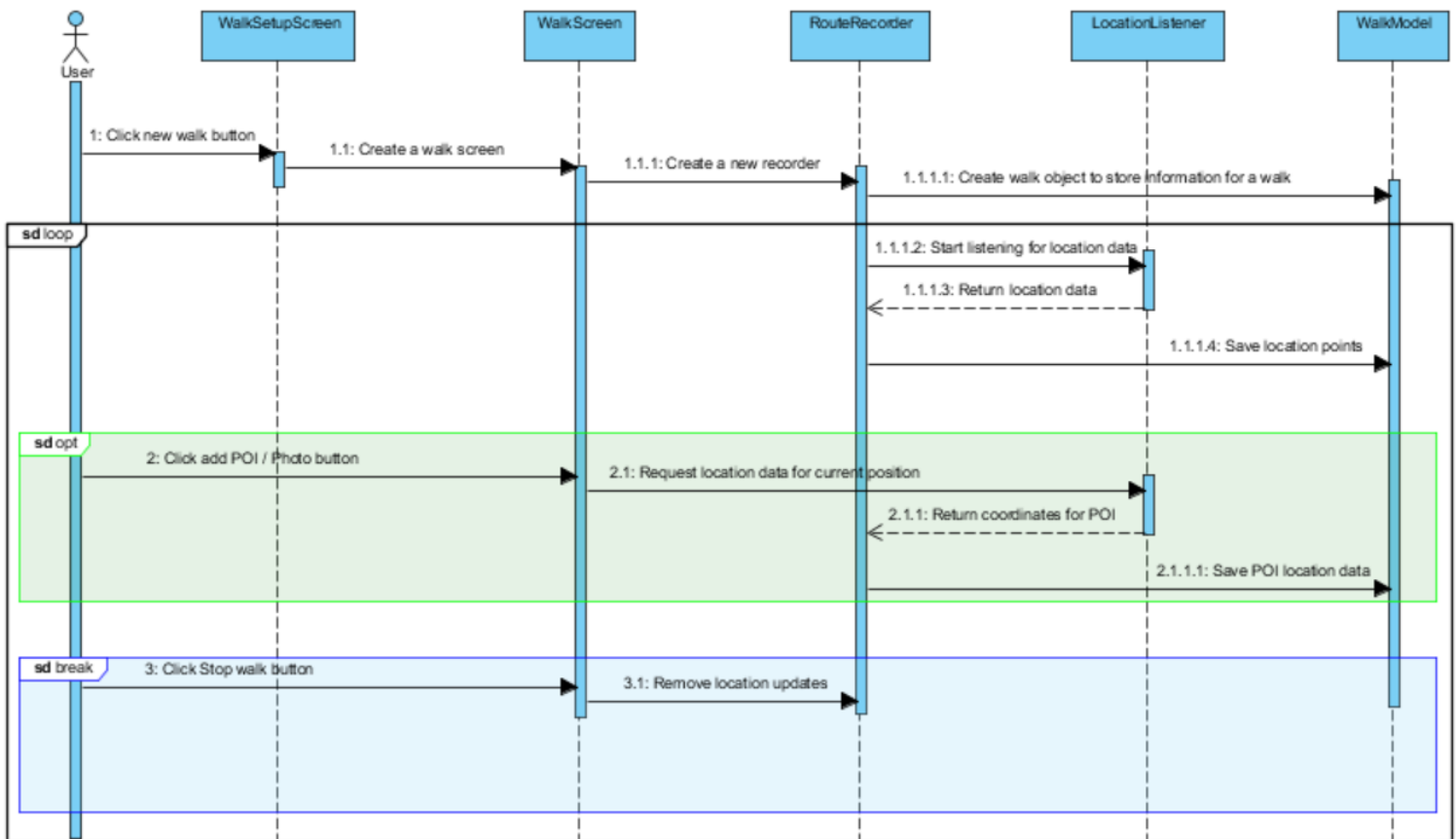


Figure 3: Android Sequence Diagram

The sequence diagram describes the recording of a walk and how the classes which are involved in the process interact. 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 brakes 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

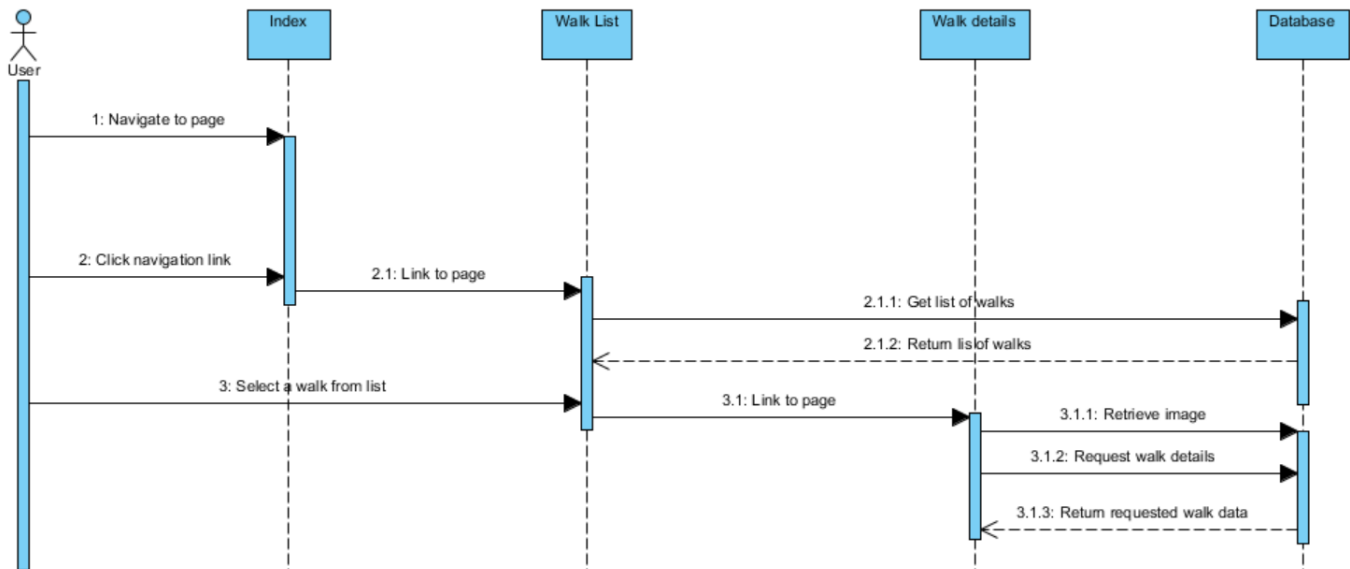
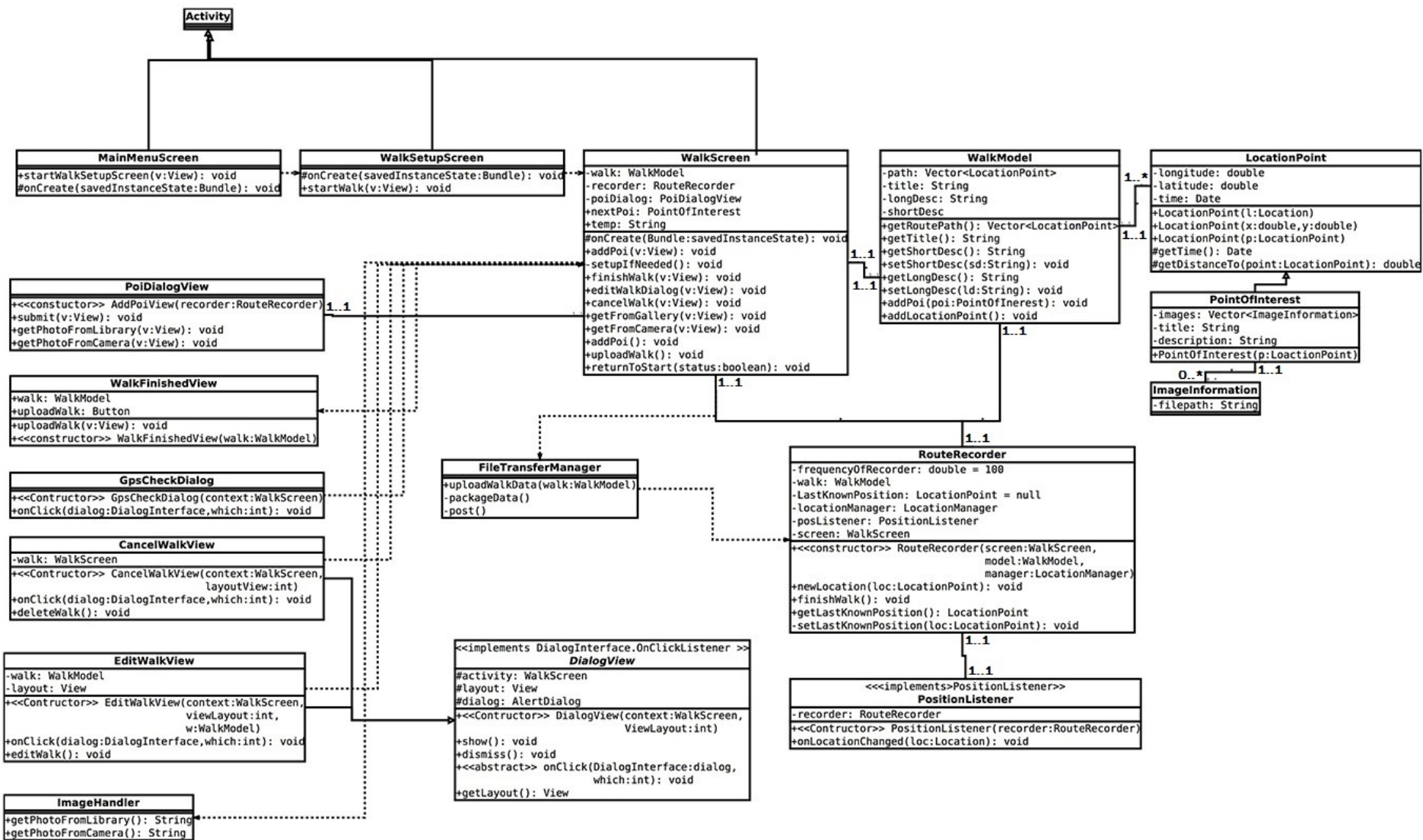


Figure 4: Web Sequence Diagram

5.1.3 Overall Interaction Sequence Diagram

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. They 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 to the user. They are the utility classes that are used by others.

5.3 Significant Algorithms

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 > X then
        add new location to walkModel
    end if
end while
```

5.3.2 PHP Algorithms

5.3.2.1 Connect To The Database

```
/**
 * This is the function to connect to the a database
 */
connectToDatabase();

/**
 * This code will connect to our own database with our
 * database name,
 * username and password
 */
```

```

$con=mysqli_connect("db.dcs.aber.ac.uk",
    cs g p 0 7 _ 1 3 _ 1 4 , "csadmgp07", "c54admgp07");

/*
 * If the php fails to connect to the database this will
   appear
 */
//heck connection
if(mysqli_connect_errno())
{
    echo "Failed to connect to MySQL: " .
        mysqli_connect_error();
}
mysqli_close($con);

```

5.3.2.2 Append To The Server Database

```

$sql = "INSERT INTO List_of_Walks(title , shortDesc , longDesc , hours , distance)
VALUES(' $title ', ' $short_desc ', ' $long_desc ', ' $hours ', ' $distance ')" ;

$queryWalk = mysqli_query($walk_conn,$sql);

$walkID = mysqli_insert_id($walk_conn);

foreach($route as $loc){

    loop through every location

    $longitude = $loc['longitude'];

    $latitude = $loc['latitude'];

    $time = $loc['time'];

    $sql = "INSERT INTO Location(walkID, latitude , longitude , timestamp)VALUES
    ( ' $walkID ', ' $latitude ', ' $longitude ', ' $time ')" ;

    $queryLoc = mysqli_query($walk_conn,$sql);

    $locID = mysqli_insert_id($walk_conn);
    /*If description , location is point of interest*/
    if(isSet($loc['description'])) {

```

```

    $description = $loc['description'];
    $name = $loc['title'];
    $sql = "INSERT INTO Place_description(description , locationId , name
        ) values(' $description ', ' $locID ', ' $name') ";
    $queryPlace = mysqli_query($walk_conn,$sql);
    $placeId = mysqli_insert_id($walk_conn);
        if(isset($loc['images']))\{
            $photoCount = 0;
            /*If there are images decode and rename them*/
            foreach($loc['images'] as $image)\{

                $image = implode($image);

                $image = str_replace("\\", "", $image,
                    $count);

                $image = base64_decode($image);

                $photoName = $walkID . "_" . $locID . "_" .
                    $placeId . "_" . $photoCount;

                file_put_contents("images/" . $photoName . ".
                    jpg", $image);

                $photoCount++;

                $sql = "INSERT INTO Photo(photoName,
                    placeId) values(' $photoName ', ' $placeId ' )
                    ";

                $queryPhoto =mysqli_query($walk_conn,$sql);

            }

        }
    }

```

5.4 TODO CHECK THE ALGORITHMS

5.5 Significant Data Structures

5.5.1 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.

5.5.2 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.

5.5.3 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. 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 -all names-, 1.8 (Release). 6th November 2013

7 DOCUMENT HISTORY

Version	CFF No.	Date	Section Changed From Previous Version	Changed by
1.0	N/A	28/11/13	Created original document	HFB1
1.1	N/A	01/12/13	Added sections created by other members. Updated config reference Updated layout.	MDA
1.2	N/A	01/12/13	Fixed some formatting issues, added information to what fields will be used.	RYG1
1.3	N/A	04/12/13	Added a new sequence diagram and a description for section 1.2	MVZ
1.4	N/A	05/12/13	Updated section 1.1. Added descriptions to all sections	MDA
1.5	N/A	05/12/13	Added a sequence diagram for the web.	MRP2
1.6	N/A	05/12/13	Updated class diagram, added FileTransferManager interface.	HFB1
1.7	N/A	06/12/13	Added Apache HTTP Client description.	JAR39
1.8	N/A	06/12/13	Added methods to the MapView interface.	HFB1
1.9	N/A	06/12/13	Changed sequence diagram for web and added overall interaction sequence diagram	MVZ
2.0	N/A	06/12/13	Updated the Introduction section.	JAR39, MRP2
2.1	N/A	06/12/13	Added web app diagram and Significant algorithms	ZAL
2.2	N/A	06/12/13	Updated author list. Updated formatting. Merged different versions of the document	MDA
2.3	N/A	06/12/13	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	06/12/13	Formatting corrections. Changed version	MDA
2.5	N/A	12/02/14	Re-wrote in LaTeX, removing feature creep	RYG1

2.6	N/A	13/02/14	Minor error checks and removal	MDA
2.7	N/A	13/02/14	Added Images	RYG1
2.8	N/A	13/02/14	Added Sequence Diagrams	RYG1