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_DS_01

Date February 17, 2014

Version 2.9

Status Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

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

2.2.1.4 WalkScreen

This is the main class, which creates the RouteRecorder. All the data handling for a walk is completed in this Activity. The method for image handling are also called within this class as well as the pop-ups; these are created within the same activity. Adding the points of interest along a walk are also used in this Activity. The uploading a walk will also be called from this Activity too.

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	WalkScreen, WalkSetupScreen, MainMenuScreen, CancelWallkView
FR2	WalkScreen, RouteRecorder, WalkModel
FR3	LocationPoint, PointOfInterest
FR4	LocationPoint, PointOfInterest
FR5	CancelWalkView, EditWalkView
FR6	WalkManager, FileTransferManager
FR7	RouteRecorder

3 DEPENDENCY DESCRIPTIONS

3.1 Component Diagrams

3.1.1 Android

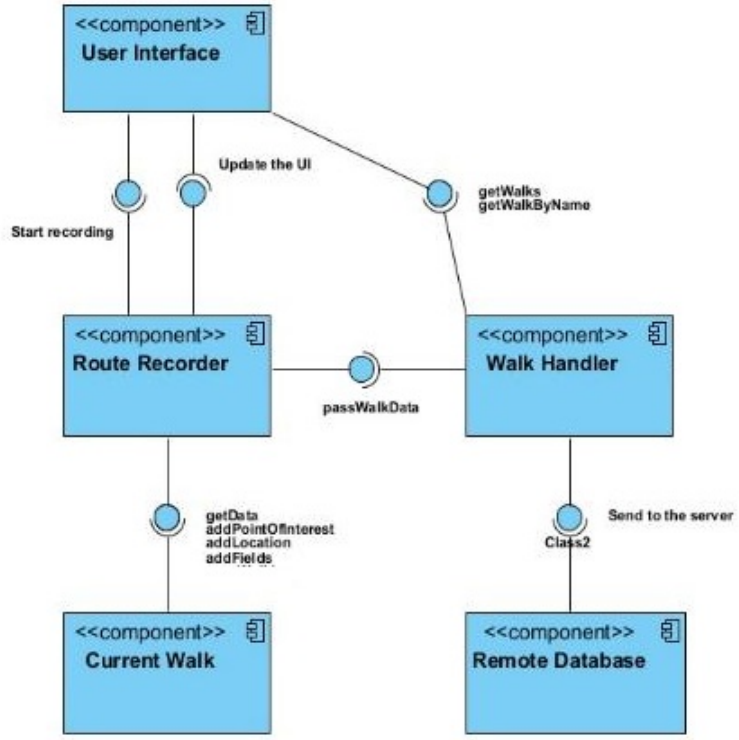


Figure 1: Android Dependency Diagram

3.1.2 Website

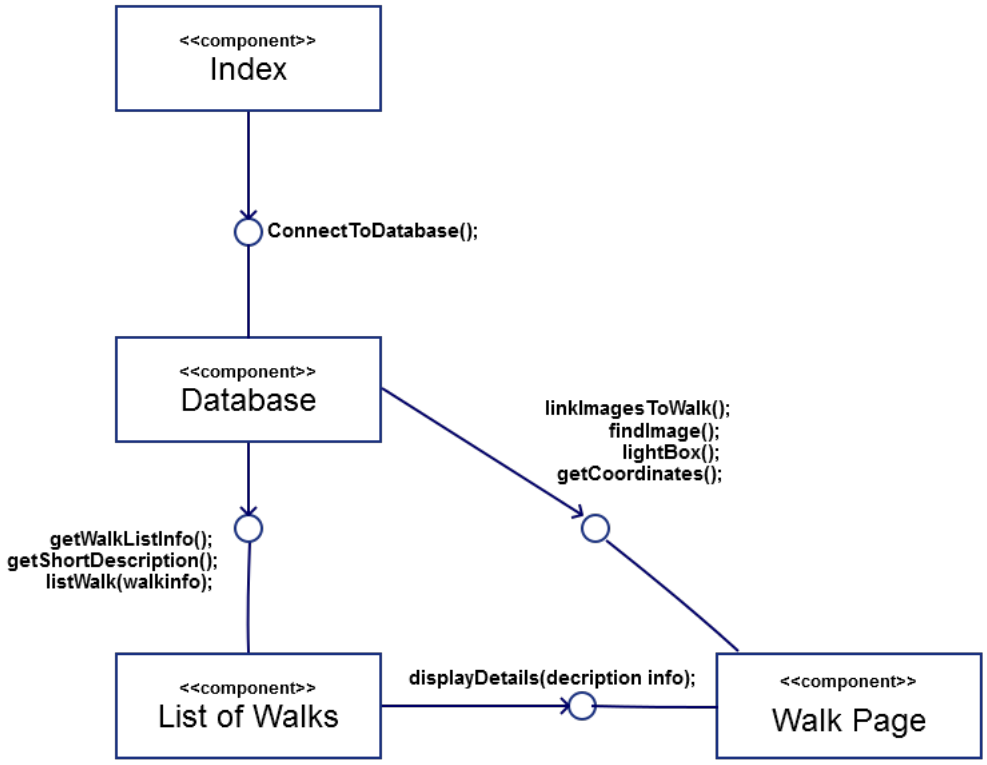


Figure 2: Web Component Diagram

3.1.3 Database Server

4 INTERFACE DESCRIPTION

This section contains method and class declarations for all major aspects of the program. The following code describes sthe public methods that are used in the program.

4.1 Screens

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

4.1.1 MainMenuScreen

```
/**
 * This class is responsible for displaying the main menu
 * screen, and reacting
 * to button presses. It is the first screen that is
 * presented to the user.
 * /
public class MainMenu extends Activity{

/**
 * Starts a new WalkSetupScreen activity, and displays it
 * to the user. It is
 * called when the user presses the start walk button.
 * @param v is the View that is called the method.
 */
public void startWalkSetupScreen(View v);
}
```

4.1.2 WalkSetupScreen

```
/**
 * This class is responsible for displaying the walk setup
 * screen which sets the
 * title and descriptions for a walk
 */
public class WalkSetupScreen  extends Activity{

   /**
    * Starts a new WalkScreen activity, displays it to the
    * user and starts
    * recording GPS data. The details that the user has
    * input, are passed to the
    * new activity.
    * @param v  is the View that called this method
    */
   public void startWalk(View v) ;
}
```

4.1.3 WalkScreen

```
/**
 * This class is responsible for displaying the walk screen
 * and reacting to button presses.
 */
public class WalkScreen extends extends Activity {

    /**
     * creates and displays a AddPoiView.
     * @param v , is the object that called the method.
     * @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 EditWalkView.
     * @param v, is the object that called the method.
     */
    public void editWalkDialog(View v);

    /**
     * creates and displays a CancelWalkView.
     * @param v, is the object that called the method.
     */
    public void cancelWalk(View v);

    /**
     *Opens the gallery to add a picture to the current walk
     * @param v the object that called the method
     */
    public void getFromGallery(View v);

    /** Opens the camera app to take a picture that will be
     * added to the walk.
```

```
     * @param v the object that called the method
     */
    public void getFromCamera(View v);

    /** Adds a new point of interest*/
    public void addPoi();

    /** Starts the upload of a walk to the server.*/
    public void uploadWalk();

    /**
     * returns the user to the start screen, is called
     * either after the upload has
     * finished or when the user cancels the walk.
     * @param status,
     * */
    public void returnToStart(boolean status);
}
```

4.2 Views

Views are subsections of the screen. Here we use popup windows to both inform the user and prompt user input.

4.2.1 DialogView

```
/**
 * An abstract class to easily create different
 * popup screens which can be
 * customzied. Implements the OnClickListener interface to
 * repsond to key
 * presses.
 */
public abstract class DialogView implements OnClickListener
    {
    /** Destroys the popup */
    public void dismiss();

    /** Displays the popup */
    public void show() ;
}
```

4.3 Models

The model classes are used to store the walk data

4.3.1 WalkModel

```
/**
 * stores all information about a single walk
 */
public class WalkModel {

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

    /**
     * Sets the name of the walk
     *
     * @param the new name of the walk.
     *
     */
    public void setTitle(String newTitle);

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

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

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

    /**
```

```
     *  @param newShortDesc, set the long description for the
         walk.
     */
    public void setLongDescription(String newLongDesc);

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

    /**
     * adds a LocationPoint to the walk.
     *
     * @param point, the location you want to add
     */
    public void addLocation(LocationPoint point);

    /**
     * works out the total distance traveled along the walk.
     *
     * @return the running total of km traveled.
     */
    public double getDistance();
    /**
     * works out the total time taken.
     *
     * @return the elapsed time since the walk was started
     */
    public double getTimeTaken();


}
```

4.3.2 LocationPoint

```
/**
 * This class stores a map position and records the time at
 * which is was taken
 * /
```

```
public class LocationPoint {
/**
 * @return the Time that the recording was made
 */
public long 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();

/**
 * works out the distance between two locations.
 *
 * @param point is the first location
 * @param point1 is the second location
 * @return the distance between the two locations in
 * kilometers
 */
public static double distBetween(LocationPoint point,
   LocationPoint point2);
}
```

4.3.3 PointOfInterest

```
/**
 * Stores information about a place of interest
 */
public class PointOfInterest extends LocationPoint{

   /**
    * adds reference to an image to the poi.
    *
    * @param newImage,is the image that is to be added
```

```
     */
    public void addImage(ImageInformation newImage);

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

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

    /**
     * @param desc, sets the description of this point.
     */
    public void setDescription(String desc);

    /**
     * gets the title of the POI
     *
     * @return String the title of the walk
     */
    public String getTitle();

    /** sets the title of this point.
     *
     * @param title, new title
     */
    public void setTitle(String title);


}
```

4.4 Controllers

Below are the important classes that interact with the walk model.

4.4.1 RouteRecorder

```
public class RouteRecorder extends Service implements
    LocationListener{
```

```
    /**
         * Gets the last known position
     * @return LocationPoint object
     *
     */
    public LocationPoint getLastKnownPosition();

    /**
     * saves the location in the WalkModel object
     *
     * @param loc
     */
    public void newLocation(LocationPoint loc);

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

4.4.2 FileTransferManager

```
/**
 * Handles the encoding and uploading of walk data.
 */
public class FileTransferManager{

    /**
     * makes a connection to data server and uploads all
     * files belonging to the
     * given file.the return value will be zero if the
     * method succeeded without problems.
     * @param walk,the walkModel that will be sent to the
     * server
     */
    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

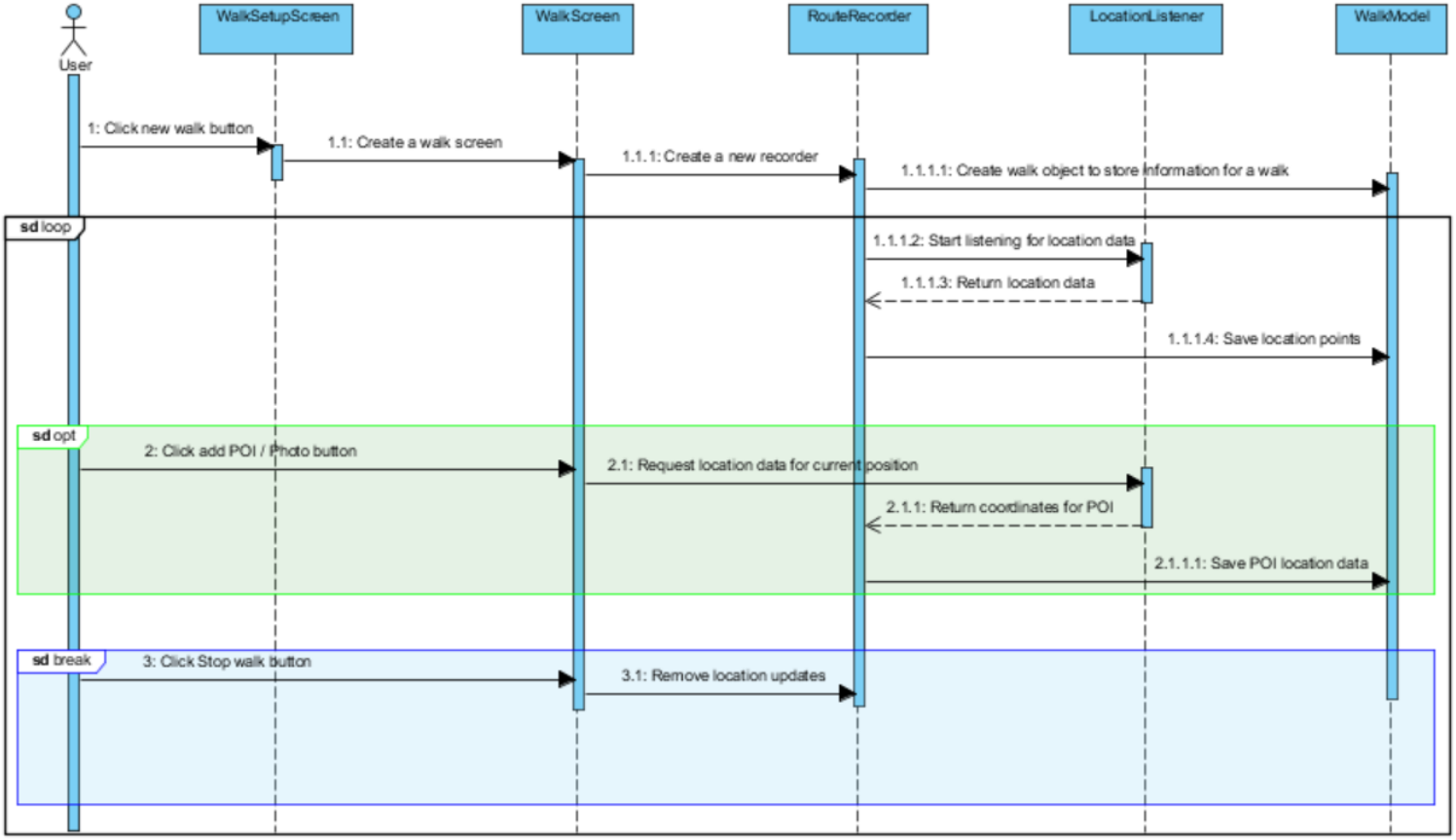


Figure 3: Android Sequence Diagram

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. 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

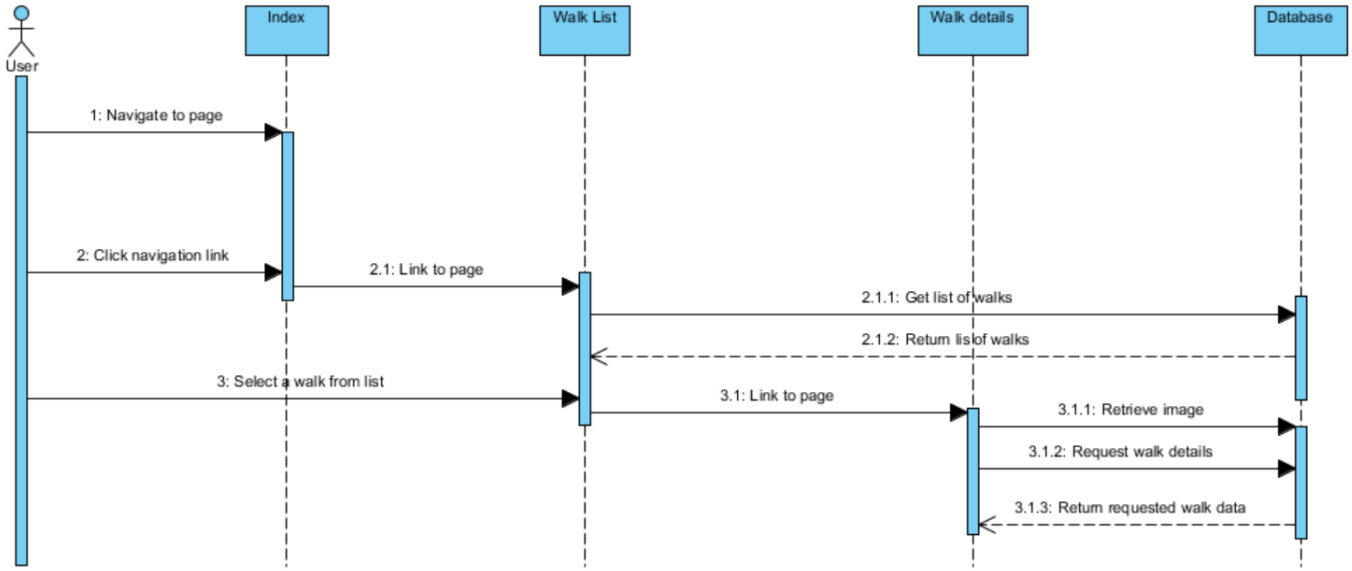


Figure 4: Web Sequence Diagram

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 In action 2, the user navigates to a page where a list of walks is displayed. The list shows all the information in the database at any one given time.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. The user can then click on a point of interest for further information about a walks location. 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

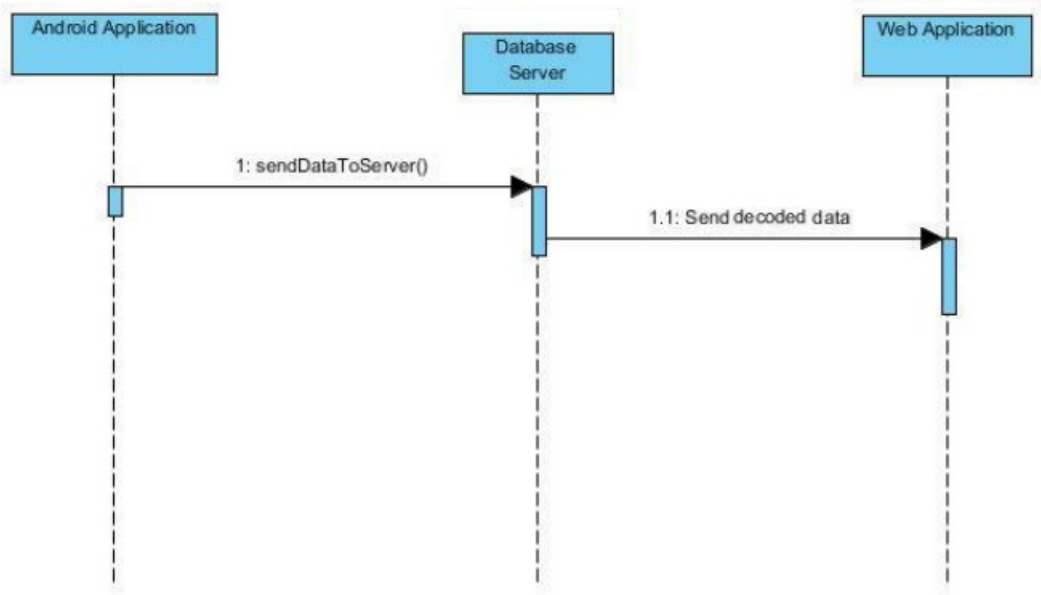
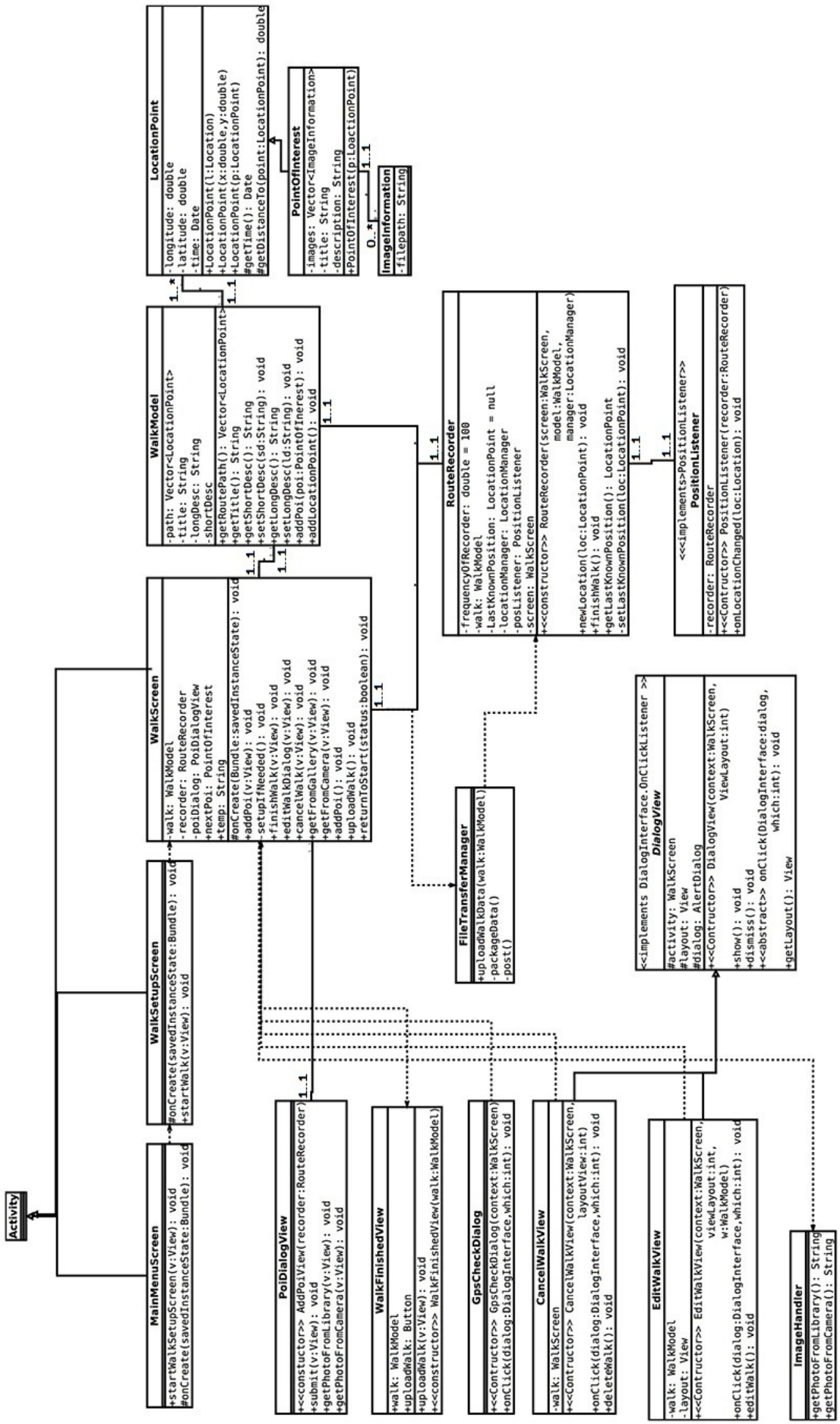


Figure 5: Overall Sequence Diagram

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 via a post request in the form of a JSON file. In action 2 the HTTP Protocol, via the Apache HTTP Client, encodes the data for sending to the database, this is done via a JSON file as a MIME type, the image is encoded as a based 64 string. In action 3 The Data Layer decodes the JSON from the Android and stores it in the database. On request theData 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


Activity
MainMenuScreen
+startWalkSetupScreen(v:View): void
#onCreate(savedInstanceState:Bundle): void
WalkSetupScreen
#onCreate(savedInstanceState:Bundle): void
+startWalk(v:View): void
WalkScreen
-walk: WalkModel
-recorder: RouteRecorder
-poiDialog: PoiDialogView
+nextPoi: PointOfInterest
+temp: String
#onCreate(Bundle:savedInstanceState): void
+addPoi(v:View): void
-setupIfNeeded(): void
+finishWalk(v:View): void
+editWalkDialog(v:View): void
+cancelWalk(v:View): void
+getFromGallery(v:View): void
+getFromCamera(v:View): void
+addPoi(): void
+uploadWalk(): void
+returnToStart(status:boolean): void
WalkModel
-path: Vector<LocationPoint>
-title: String
-longDesc: String
-shortDesc
+getRoutePath(): Vector<LocationPoint>
+getTitle(): String
+getShortDesc(): String
+setShortDesc(sd:String): void
+getLongDesc(): String
+setLongDesc(ld:String): void
+addPoi(poi:PointOfInerest): void
+addLocationPoint(): void
LocationPoint
-longitude: double
-latitude: double
-time: Date
+LocationPoint(l:Location)
+LocationPoint(x:double,y:double)
+LocationPoint(p:LocationPoint)
#getTime(): Date
#getDistanceTo(point:LocationPoint): double
PointOfInterest
-images: Vector<ImageInformation>
-title: String
-description: String
+PointOfInterest(p:LoactionPoint)
ImageInformation
-filepath: String
RouteRecorder
-frequencyOfRecorder: double = 100
-walk: WalkModel
-LastKnownPosition: LocationPoint = null
-locationManager: LocationManager
-postlistener: PositionListener
-screen: WalkScreen
+<<constructor>> RouteRecorder(screen:WalkScreen,
model:WalkModel,
manager:LocationManager)
+newLocation(loc:LocationPoint): void
+finishWalk(): void
+getLastKnownPosition(): LocationPoint
-setLastKnownPosition(loc:LocationPoint): void
<<implements>>PositionListener>>
PositionListener
-recorder: RouteRecorder
+<<Contructor>> PositionListener(recorder:RouteRecorder)
+onLocationChanged(loc:Location): void
FileTransferManager
+uploadWalkData(walk:WalkModel)
-packageData()
-post()
PoiDialogView
+<<constuctor>> AddPoiView(recorder:RouteRecorder)
+submit(v:View): void
+getPhotoFromLibrary(v:View): void
+getPhotoFromCamera(v:View): void
WalkFinishedView
+walk: WalkModel
+uploadWalk: Button
+uploadWalk(v:View): void
+<<constructor>> WalkFinishedView(walk:WalkModel)
GpsCheckDialog
+<<Contructor>> GpsCheckDialog(context:WalkScreen)
+onClick(dialog:DialogInterface,which:int): void
CancelWalkView
-walk: WalkScreen
+<<Contructor>> CancelWalkView(context:WalkScreen,
layoutView:int)
+onClick(dialog:DialogInterface,which:int): void
+deleteWalk(): void
<<implements DialogInterface.OnClickListener >>
DialogView
#activity: WalkScreen
#layout: View
#dialog: AlertDialog
+<<Contructor>> DialogView(context:WalkScreen,
ViewLayout:int)
+show(): void
+dismiss(): void
+<<abstract>> onClick(DialogInterface:dialog,
which:int): void
+getLayout(): View
EditWalkView
-walk: WalkModel
-layout: View
+<<Contructor>> EditWalkView(context:WalkScreen,
viewLayout:int,
w:WalkModel)
+onClick(dialog:DialogInterface,which:int): void
+editWalk(): void
ImageHandler
+getPhotoFromLibrary(): String
+getPhotoFromCamera(): String
1..1
1..*
0..*

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 dont use the whole screen and do not change the displayed screen only create new Views. The classes WalkModel,PointOfInterest and LocationPoint 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 than X then
        add new location to walkModel
    end if
end while
```

5.3.1.2 JSON Encoder

```
json.add("title", data.getTitle )
json.add("short_desc", data.getShortDescription )
json.add("long_desc", data.getLongDescription )
json.add("hours", data.getTimeTaken )
json.add("distance", data.getDistance )
for  all data getPath as point do
    locationObject.add("longitude", point.getLongitude )
    locationObject.add("latitude", point.getLatitude )
    locationObject.add("time", point.getTime )
    if point is an instance of PointOfInterest then
        locationObject.add("description",point.getDescription )
        locationObject.add("title", point.getTitle )
        for all point.getImages as image do
            imagObject.add ("file_data", image.getImageAsString )
            imageArray.add imagObject
```

```
            location.put("images", images)
        end for
    end if
    pointsArray.put(locationObject)
end for
json.add("route", pointsArray)
```

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", "csgp07_13_14","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

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

mysqli_query($walk_conn,$sql);
$locID = mysqli_insert_id($walk_conn);
if(isSet($loc['description'])){
    $description = $loc['description'];
    $name = $loc['title'];
    $sql = "INSERT INTO Place_description(description, locationId, name)values('$description', '$locID', '$name')";
    mysqli_query($walk_conn,$sql);
    mysqli_insert_id($walk_conn);
            if(isSet($loc['images'])\{
                $photoCount = 0;
                foreach($loc['images']as $image)\{
                    $image = implode($image);
                    $image =  base64_decode($image);
                    $photoName = $walkID . "_" . $locID . "_" . $placeId . "_" . $photoCount;

                    file_put_contents("images/".$photoName . ".jpg",$image);
                    $photoCount++;
                    $sql = "INSERT INTO Photo(photoName, placeId)VALUES('$photoName', '$placeId')";
                    mysqli_query($walk_conn,$sql);

                    }
                }
            }
```

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. 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 Enginerring 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, R. Gouldsmith and others 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
2.9	N/A	13/02/14	Updated database file saver algorithm	MDA