CS211 Group 15
Design Specification

Authors: lpd1; jcm14; jap38; jaj42; owd2; jam67; gad16
Date: 02-12-2013
Version: 0.1
Config Ref: DOC-Des-DesSpec-001
Date: 03-12-13

Department Of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

Contents

1 INTRODUCTION

1.1 Purpose of this document

The purpose of this document is to show the design specification for the Walking Tour Creator.

1.2 Scope

This document will include a decomposition description which will include programs in the system, significant classes in each program, modules shared between programmes and mapping from requirements to classes; a dependency description which will include component diagrams for all programs and inheritance relationships; an interface description; and a detailed design of the Walking Tour Creator which will include sequence diagrams, significant algorithms and significant data structures.

1.3 Objectives

- Describe each of the programs and the relationship between programs
- Provide a short description of the purpose of each class
- Describe the relationships and dependencies between modules
- Provide component diagrams for each program, showing the method links between modules
- Produce an interface description which includes:
  - The name and type of the class or interface
  - Classes or interfaces which it extends
  - Public methods implemented by the class or interface, including parameter names and types for each method
- Provide sequence diagrams which shows how the classes work together for the major operations of the program
- Provide a textual description of difficult parts of the system that need to be implemented (algorithms)
- Outline significant data structures using class diagrams to show entity relationships between classes, along with object diagrams which show how static relationships in class diagrams work types for each method

2 DECOMPOSITION DESCRIPTION

2.1 Programs In System

We can split the overall system down into two programs, which allow the system to be used, and the database to be accessed, both on the internet and on any Android based mobile system.

- The mobile application written in Java for Android
- Web-based application to allow a user to view submitted tours

While these two programs will be handled separately from each other, they will share certain properties, such as similar data structures to handle the same type of data, and both will use the same database to store and retrieve data
The Server is PHP based, and handles data to and from both of the programs and the SQL Database.
We will be using a PHP server to process incoming and outgoing database requests from both the web application and the mobile application.

2.2 Significant Classes In Each Program

2.2.1 Significant classes in Android Application

Qualities: The Application will make use of:

- Android.App.Activity;
- Android.Location;
- Android.Net

These are the core API section that are used by the app.

Model

Public Class Model
This Object is responsible for binding all other elements together, and being a base level interface for any Activity to access Instances of Objects or Constant Values.

Public Class GPSLocation
This Object is responsible for managing the GPS Location of the mobile device, and return this information to Public Class WayPoint.

Public Class Route
This Object is responsible for holding and managing all WayPoint objects. Waypoints will be stored in a Queue Structure, from Java.Util. There will be a method to receive a specific, or all Waypoints, there will also be a method to remove and add waypoints. Any other methods will be supplementary (non-core), or inherited.

Public Class WayPoint
This Object is responsible for storing the information about each waypoint. This information includes, location from Android.Location; location timestamp from Android.Location short description in String format; long description in String format; image list using Android.Widget.ImageView; an optional sound recording using Android.Media.MediaRecorder.

Public Abstract Class AbsHTTPPostBuilder
This Object is responsible for building a String to use in the HTTP Post Request. It does not upload it, however it manages the build, and stores it during this phase. It will use Java.Lang.StringBuilder to construct this. The Post will consist of all elements from WayPoint, and a waypoint id from Route

Public Class HTTPPostSender
This Object is responsible for sending the HTTP Post. It will also manage monitoring of network availability, which controls the ability to upload information. It will run its own thread, and send when it is possible.

View

Public Class ActivityLogin
This Activity is responsible for the display of the Login screen and sending information to the Model via its Controller. It will link to the Register and Main Menu Activitys.

Public Class ActivityRegister
This Activity is responsible for the display of the Register screen and sending information to the Model via its Controller. It will link to the Login Activity.

Public Class ActivityMenu
This Activity is responsible for the display of the Main Menu screen and sending information to the Model via its Controller. It will link to the Login, Route and Settings Activities.

Public Class ActivtyRoute
This Activity is responsible for the display of the Route screen and sending information to the Model via its Controller. It will link to the WayPoint, Settings, and Menu Activitys.

Public Class ActivityWayPoint
This Activity is responsible for the display of the WayPoint screen and sending information to the Model via its Controller. It will link to the Route, Settings, and Menu Activities.

Public Class ActivitySettings
This Activity is responsible for the display of the Settings screen and sending information to the Model via its Controller. It will link to the Route, WayPoint, Menu, Login Activitys.

2.3 Modules Shared Between Programs

2.3.1 Database Elements

The Database will be using MySQL engine. All parties will interact through this Database. The Mobile Application, will have an inhouse local database, to store waypoint, and walk information, while they cannot be uploaded. The local database connection will be declared in Public Class Model.

2.3.2 HTTPPost Object

This is a formatted String Object, that will be recognised by the Web and Mobile Applications. Its design will be used in HTTPPostBuilder, HTTPPostSender, and HTTPPostReciever.

2.4 Mapping From Requirements To Classes

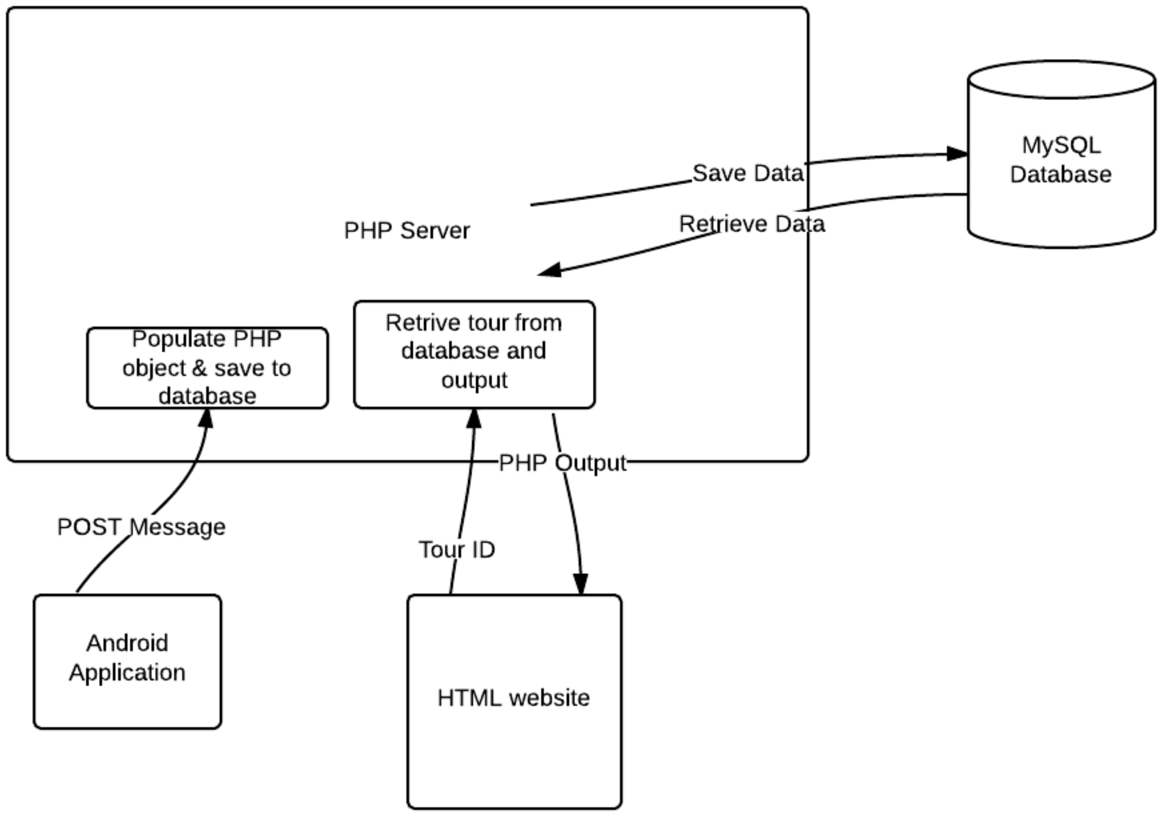
The Classes that actively translate to fulfilling the Functional Requirements are mainly located within the Android application, although the WTD will be required to complete a few of the requirements not completed by the Android application.

FR1	This will be fulfilled by the Android application. This is the most basic functional requirement, and will most likely be partly completed by [Main Class] as opposed to any specific class. Creating a new walk will be processed by the [Walk Creater] and will have the information sent to the server for later recall by either the App or the Website.
FR2	The Title and the descriptions of the walk will be stored as properties of the Walk, to be displayed upon access by either the Application or the Website. This will be done by displaying the appropriate properties when the file is accessed through the [Opening Class]. The Application will use the [Walk Creator] to save the properties when the walk is initially created by the user.
FR3	Adding the Process will be done through very similar methods as used through saving the Title and Descriptions, except that the Waypoint will be saved as an object with the GPS Co-Ordinates, Name, Description and Time Stamp saved as properties, and will be saved itself to the walk through the [Walk Creator]
FR4	This functional requirement will be completed using the same class as FR3, as the photo will also be saved as an object with different properties. The properties of the image will be the image file path, GPS Co-Ordinates, Name, Description and Time Stamp.
FR5	Within the Walk Creator their will be a button to Delete the currently in progress walk, which will link to [Deletion Class?] which will permanently delete that walk, along with all of it's properties and objects associated uniquely with that walk. Due to the fact that this could be disastrous for the user their will be a confirmation message displayed as part of the running of this class.
FR6	This functional requirement will be completed by the [Server Upload Class]. The information will be uploaded to the server using PHP, specifically using a HTTP POST method to upload the required information to a predefined URL, which will then be translated into the correct formatting by the server, and then uploaded into the database.
FR7	Should the user switch away from the application it will use a predefined Android local storage method to keep the user's data until such point that the user either closes the program or switches back to it.
FR8	This will be completed by the WTD. It will use the information loaded onto the server (see FR6 above) as well as a Google Maps API in order to correctly display the information required by the user on the map, correct to the corresponding GPS Co-Ordinates. This will be done by the client side web application's main displaying class, [Will the Web Site even have classes?].

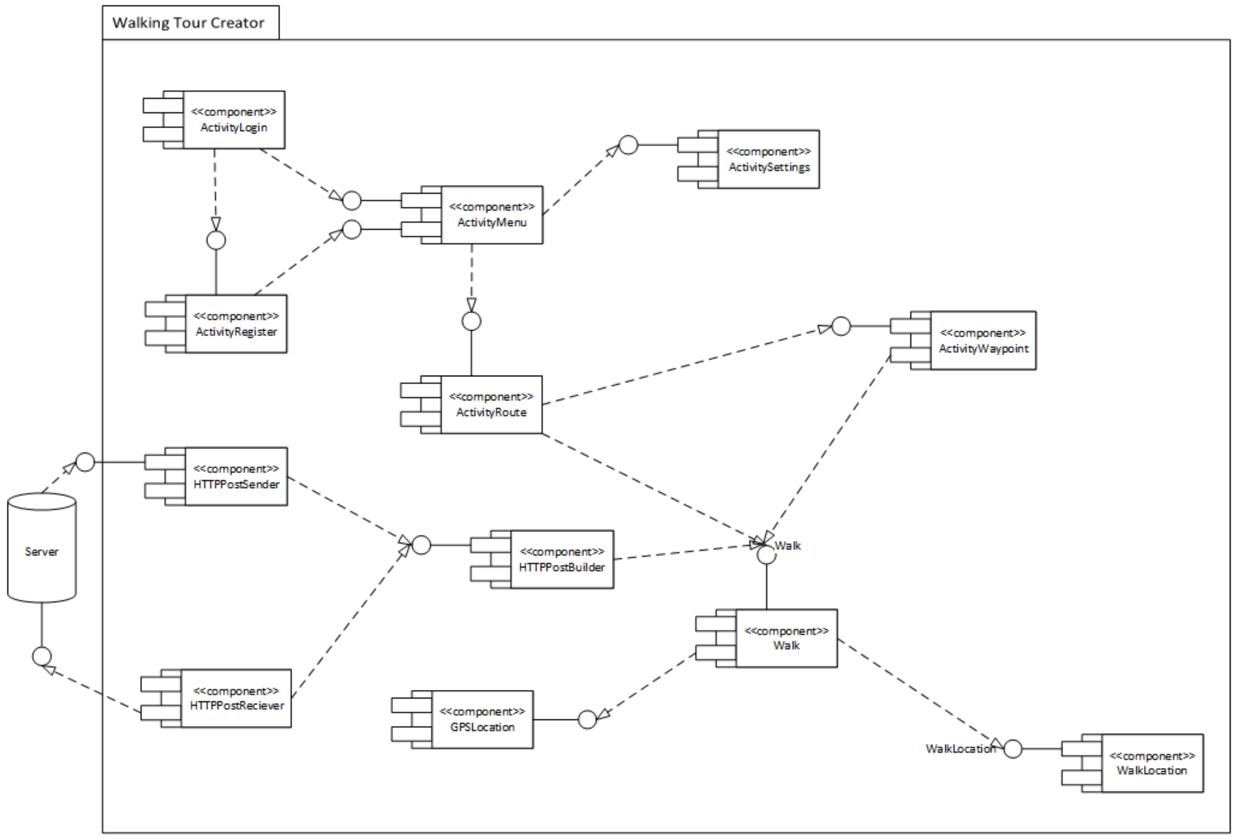
3 DEPENDENCY DESCRIPTION

3.1 Component Diagrams

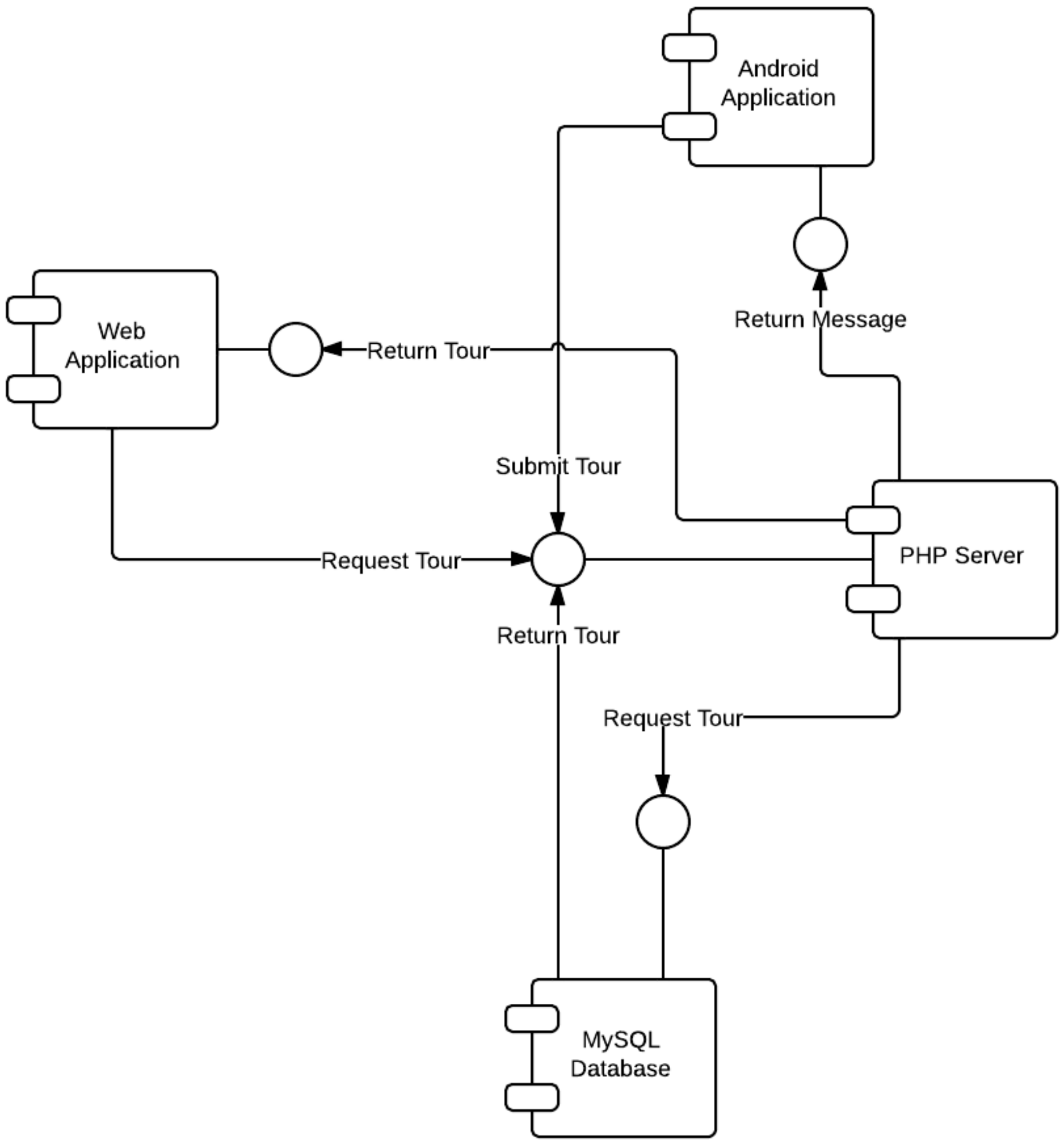
3.1.1 Deployment Diagram



3.1.2 Component diagram for Walking Tour Creator


Walking Tour Creator
<<component>> ActivityLogin
<<component>> ActivityMenu
<<component>> ActivitySettings
<<component>> ActivityRegister
<<component>> ActivityWaypoint
<<component>> ActivityRoute
<<component>> HTTPPostSender
Server
<<component>> HTTPPostBuilder
Walk
<<component>> Walk
<<component>> HTTPPostReciever
<<component>> GPSLocation
WalkLocation
<<component>> WalkLocation

3.1.3 Component diagram for Walking Tour Displayer



4 INTERFACE DESCRIPTION

4.1 Class 1 Interface Specification

HTTPPostBuilder:

```java
package uk.ac.aber.cs22120.fuzzyNinja.pathFinder;

import java.util.StringBuilder;

/** This class is responsible for building a string to use in an
    HTTP post request
* It is only responsible for building and storing the request
    string.
*/
public class HTTPPostBuilder {
  private StringBuilder postString; // The string used in an HTTP
    post request

  /** Builds an HTTP post request based on the attributes of a Walk
      objects
  * @param postWalk the Walk object used to build the HTTP post
    request
  */
  public void buildString(Walk postWalk);
}
```

GPSLocation:

```java
package uk.ac.aber.cs22120.fuzzyNinja.pathFinder;

public class GPSLocation {
  private double latitude; // Represents a latitude, given in
    degrees
  private double longitude; // Represents a longitude, given in
    degrees

  /** Sets the latitude
  * @param lat the latitude in degrees
  */
  public void setLatitude(double lat);

  /** Gets the latitude
  * @return Returns the latitude in degrees
  */
  public double getLatitude();

  /** Sets the longitude
  * @param lng the longitude in degrees
  */
  public void setLongitude(double lng);

  /** Gets the longitude
  * @return Returns the longitude in degrees
  */
  public double getLongitude();
}
```

HTTPPostSender:

```java
package uk.ac.aber.cs22120.fuzzyNinja.pathFinder;

import java.lang.Thread;
import android.net.ConnectivityManager;
/**
* This class is responsible for sending the HTTP Post.
* It will also manage monitoring of network availability,
* which controls the ability to upload information.
* It will run its own thread, and send when it is possible.
*/
public class HTTPPostSender implements Runnable{
  Thread networkThread;
  boolean isAvailable;
  ConnectivityManager cm;
  /** Sends an HTTP post request to the web server
  * @param request the string obtained from HTTPPostBuilder
  */
  public void send(String request);
}
```

Walk

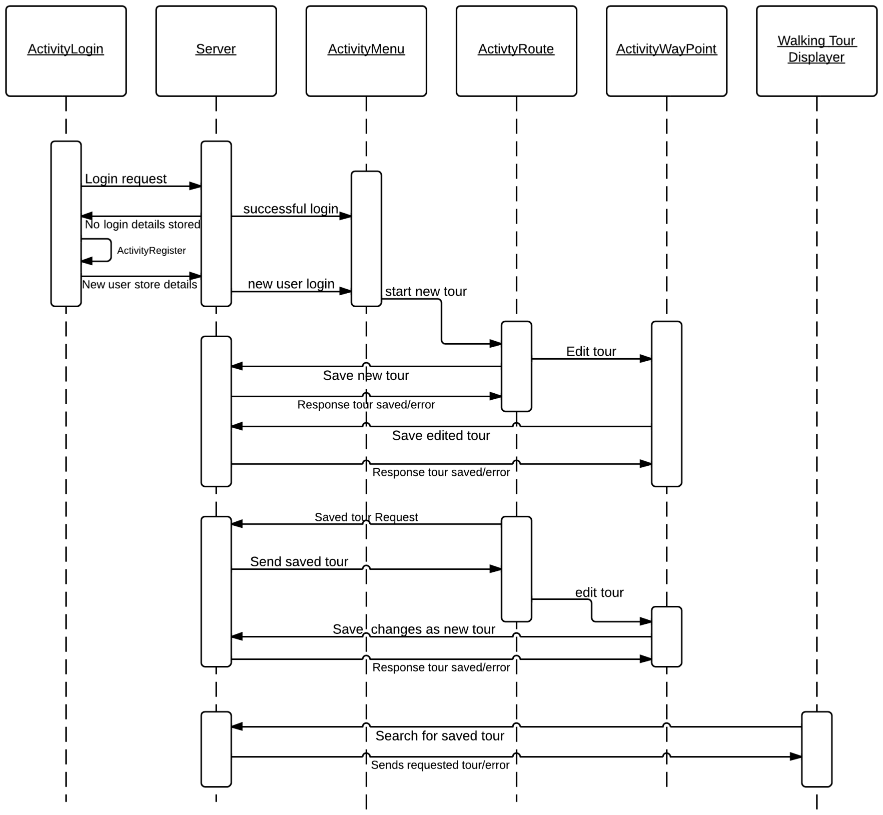
```java
package uk.ac.aber.cs22120.fuzzyNinja.pathFinder;

import uk.ac.aber.cs22120.fuzzyNinja.pathFinder.WalkLocation;
import java.util.ArrayList;

/**
* This class contains information associated with a walking tour.
*
*/
public class Walk {
  private String title; // The title of the walking tour
  private String shortDescription; // A short description of the
    walking tour
  private String longDescription; // A long description of the
    walking tour
  private ArrayList<WalkLocation> waypoints; // A list of the
    waypoints along the walking tour

  /** Sets the title of a given walking tour
  * @param title the title of the walking tour
  */
  public void setTitle(String title);

  /** Gets the title of the walking tour
  * @return Returns the title of the walking tour
  */
  public String getTitle();

  /** Sets the short description of the walking tour
  * @param description the short description for the walking tour
  */
  public void setShort(String description);

  /** Gets the short description of the walking tour
  * @return Returns the short description for the walking tour
  */
  public String getShort();

  /** Sets the long description of the walking tour
  * @param description the long description for this walking tour
  */
  public void setLong(String description);

  /** Gets the long description of the walking tour
  * @return Returns the long description for the walking tour
  */
  public String getLong();

  /** Adds a location to the end of the list
  * @param location the location added to the list
  */
  public void addLocation(WalkLocation location);

  /** Gets a location from the queue at the specified index
  * @param index the index from which to get the location
  */
  public WalkLocation getLocation(int index);

  /** Deletes a the location at the specified index
  * @param index at which to delete location
  */
  public void deleteLocation(int index);

  /**Default constructor*/
  public Walk() {}
}
```

WalkLocation:

```java
package uk.ac.aber.cs22120.fuzzyNinja.pathFinder;

import uk.ac.aber.cs22120.fuzzyNinja.pathFinder.GPSLocation;
import java.util.ArrayList;
import java.util.Date;

/** Specifies a waypoint in the walking tour
*
*/
public class WalkLocation {
  private GPSLocation coordinates; // GPS coordinates for the
    waypoint
  private String name; // Name given to the waypoint
  private String description; // Description given to the waypoint
  private ArrayList<Bitmap> photos; // List of photos associated
    with the waypoint
  private Date timestamp; // The date and time the waypoint was
    recorded

  /** Default constructor for this class
  * GPS coordinates and timestamp should be assigned in this method
  */
  public WalkLocation();

  /** Adds a Bitmap object to the list of photos
  * @param photo photo to be added
  */
  public void addToPhotos(Bitmap photo);

  /** Gets a Bitmap object from list of photos
  * @param index index of the required Bitmap object
  * @return Returns the Bitmap object at the specified index
  */
  public Bitmap getFromPhotos(int index);

  /** Deletes an element at the specified index
  * @param index index of the Bitmap to be deleted
  */
  public void removeFromPhotos(int index);
}
```

5 DETAILED DESIGN

5.1 Sequence Diagrams



Above is the sequence diagram showing how the mobile application works and the messages passed through the mobile application, the walking tour displayer (website) and the server. The six objects show the mobile application screens (ActivityLogin , ActivityMenu , ActivtyRoute and the ActivityWayPoint), the server and the walking tour displayer to view the tours. The arrows between them show the messages sent between the objects. This chart is useful for showing the processes of the system, how the server interacts with both the website and mobile application and the ordered way the classes interact with each other.

5.2 Significant Algorithms

In this section we have designed pseudo code outlining each of the major algorithms in our system.

5.2.1 Algorithm 1 - Creating A New Walk

Classes Used:
GPSLocation
Walk
WalkLocation

```
while App is open do
  if User presses "Create new walk screen" then
    if User has connection to internet and User has GPS signal then
      Show Walking Tour Creation screen
    else
      Show error message informing the user that they must be connected
      to create a walking tour
    end if
  end if
end while
```

This algorithm allows the user to create a new walk on the Walking Tour Creator application. It checks whether the user is connected to the internet and has a GPS signal before allowing them to begin the process.

5.2.2 Algorithm 2 - Adding Waypoints To The Walk

Classes Used:
GPSLocation
Walk
WalkLocation

```
while App is open do
  if User is in walking tour creation mode and User has internet and GPS
  connection and User presses "Add Waypoint" then
    Take current GPS Location
    Take details of waypoint
    Save Waypoint information to local database
  end if
end while
```

This algorithm will allow the user to add a new waypoint to the walk. When the user presses the "add waypoint button" it will prompt for details of the waypoint - such as images, description, title etc. and save them to the local database.

5.2.3 Algorithm 3 - Sending The Walk To The Server

Classes Used:
HTTPPostSender

```
if User presses submit button and User has internet and GPS connection then
  Get tour information from local Android database
  Create instance of post structure as defined in Section 5.3.$x$
  for all Waypoints in Database do
    Add waypoint information to POST
  end for
  Send POST to server
  Wait for server response
  if No response message received or Message shows that an error has occurred then
    Show error message to user
  else
    Show success message to user
  end if
end if
```

This algorithm details the process of sending a walk to the server, using a HTTP POST message containing the information for the Walk and each WalkLocation.

5.2.4 Algorithm 4 - Processing The Walk On The Server

```
Receive POST from mobile application
Check structure of POST
if POST is structured correctly then
  Create database entry for tour in tours table
  return Tour ID
  if Tour ID returned then
    for all Waypoints in tour do
      Create database entry for waypoint in tours table, referencing tour ID
    end for
    Return success message to mobile application.
  else
    Return error message to mobile application.
  end if
else
  Return error message to mobile application.
end if
```

This is the algorithm to process the walk, submitted via POST, on the server. It checks that the POST is structured correctly, and then iterates through each WalkLocation and adds it to the database.

5.3 Significant Data Structures

The data structures that will mainly be used on the Android application are ArrayLists. The reason ArrayList has been chosen over an Array is because, in Java an Array is a fixed length data structure whereas an ArrayList is a variable length Collection class. This means the ArrayList will re-size itself when it reaches its capacity. This is required because the size of the arrays we will be using will not be known until after elements have been entered into it.

5.3.1 WalkLocation

An object of the WalkLocation class will hold variables which contain data about the current location the user is at, such as:

- Coordinates
    - The latitude and longitude value of the location.
- Name
    - The name given to the location by the user.
- Description
    - A description of the location given by the user.
- Photos
    - An optional image of the location selected by the user .
- Timestamp
    - The date of the current moment when the user creates the current WalkLocation object.

5.3.2 Walk

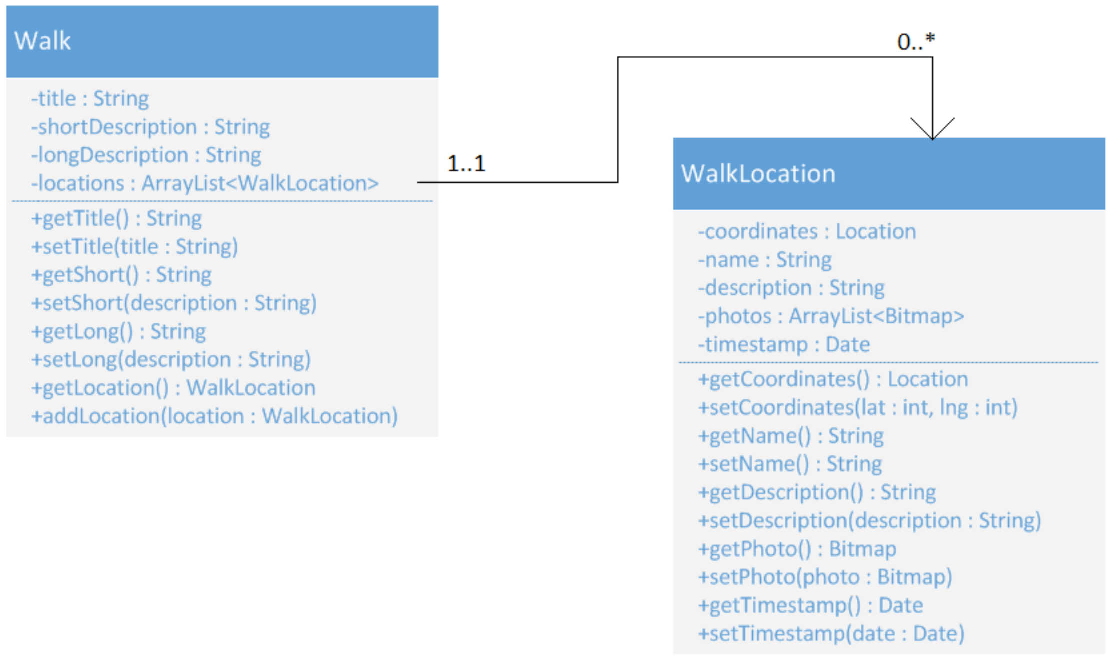
An object of the Walk class will hold variables which contain data about the walk tour the user created such as:

- Title
    - The name given to the walk tour.
- Short Description
    - A short summary of the walk tour.
- Long Description
    - A full description of the walk tour.
- Locations
    - An ArrayList of WalkLocation objects.

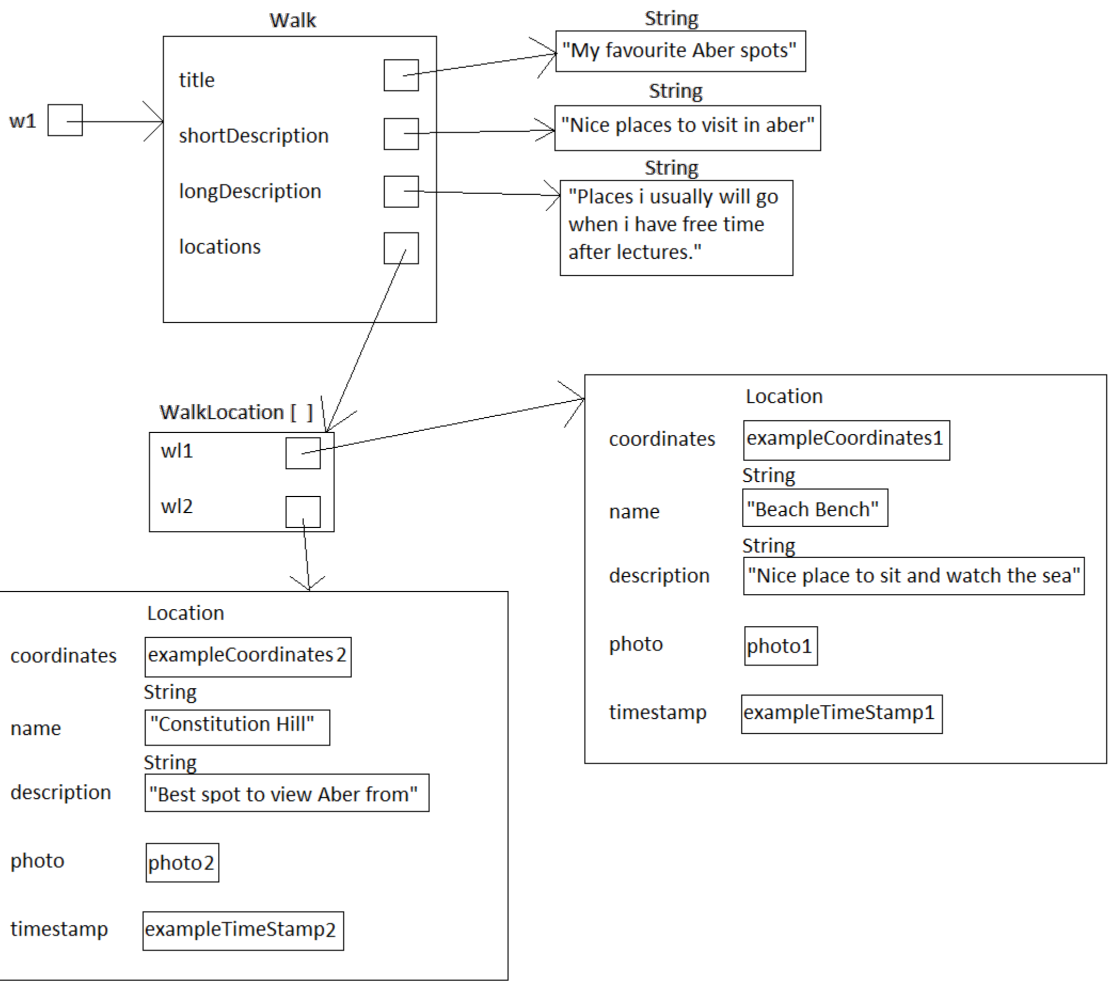
5.3.3 POST Stucture

The application will need a method of sending a walk tour to the server, when the user decides to upload their walk tour. The POST message will be sent from the android application to the PHP server; the server will then populate the PHP object and save to the MYSQL database.

Class Diagram:


Walk
-title : String
-shortDescription : String
-longDescription : String
-locations : ArrayList<WalkLocation>
+getTitle() : String
+setTitle(title : String)
+getShort() : String
+setShort(description : String)
+getLong() : String
+setLong(description : String)
+getLocation() : WalkLocation
+addLocation(location : WalkLocation)
1..1
0..*
WalkLocation
-coordinates : Location
-name : String
-description : String
-photos : ArrayList<Bitmap>
-timestamp : Date
+getCoordinates() : Location
+setCoordinates(lat : int, lng : int)
+getName() : String
+setName() : String
+getDescription() : String
+setDescription(description : String)
+getPhoto() : Bitmap
+setPhoto(photo : Bitmap)
+getTimestamp() : Date
+setTimestamp(date : Date)

Object Diagram:



6 DOCUMENT HISTORY

Version	**Date**	**Description**
0.1	03-12-13	Initial document creation.
1.0	04-12-13	First draft complete.
2.0	06-12-13	Reviewed and modified version complete