Design Specification: GROUP 12

Authors: bea16, dpb1, alh32, fik, anp28, srp11, lps1, lot15, thw10

Config Ref: SE_12_DS_01
Date: 28/01/14
Version: 1.0
Status: Final

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

Table of Contents

1 INTRODUCTION

1.1 Purpose of this Document

This document is here to ensure that the format and content of the information which will be a part of the final design specification document in the the software engineering group project. It will help to make sure that all the major details and functions of the system are described. This will include a description of the programs which are a part of the overall system with a further description of significant classes in each one of them. Component diagram will be shown. The system will be described in detail using a variety of diagrams, algorithms and data structures.

1.2 Scope

This document specifies the standards for writing software design specifications. It describes the necessary layout and content of a design specification. This document will be produced in accordance to the design specification standards document **[1]** SE.QA.05.A.

The main sections to be covered are:

- Introduction: to include the purpose of the document, scope and objectives.
- Decomposition Description: this section will include a description of the programs in system as well as significant classes in each program. A table showing Specific Requirements and how they are mapped into classes will also be provided.
- Dependency Description: this section will include a component diagram. What is more, inheritance relationships will be described.
- Interface Description: this section will include interface specification for each of the classes.
- Detailed Design: the final section will include a sequence diagram, significant algorithms, significant data structures as well as an entity relationship diagram.
- References: will include references to the external sources used when producing the document.

1.3 Objectives

The main objective is to aid the production of a design specification which is a complete and accurate translation of the client's requirements into a description of the design elements necessary for the implementation phase. These will include the software structure, components, interfaces, and data. In a complete design specification, each requirement must be traceable to one or more design entities. **[1]**

Objectives are:

- To describe the programs in system and how they will interact together
- To show the components and inheritance within the system using component diagrams and inheritance relationships.
- To specify interfaces for each program.
- To describe significant algorithms and data structures.

2 DECOMPOSITION DESCRIPTION

2.1 Programs in system

Firstly we decompose the software into 3 main subsystems which are:

- The android application.
- The server.
- The web application.

2.1.1 The Android Application (WTC)

The Walk program provides a user interface on a android platform to enable a user to make a walk, way-points, amend walk, way-point, take pictures of way-points and put in information about the way-points
Implements requirements (FR1),(FR2),(FR3),(FR4),(FR5),(FR6),(FR7),(PR1),(PR2) and (DC1) . This program will be form-based so it will comply with (EIR1). The walk program will send the information it generates from the user as described in(FR6)

2.1.2 The Server (WTD)

The Server provides a storage space for the database and all the information about walks so that they can view them. This should implement (FR9),(PR1) and (DC3). The Server will also use a Test Data file which is described in (DC2). The server will save the information it receives inside of a database.

2.1.3 The Website (WTD)

The Website provides a professional user interface for a person to view a walk online. It also allows a user to search for a specific walk. Details on each walk will then be displayed. Implementing requirements (FR8),(EIR1) and (PR1). The Website will query information from the Server and display it on to the google maps interface so the user can take the tour. The locations within the tour will be shown as pins. The line showing the tour will be drawn based on the way poins recorded during the walk.

2.2. Significant classes in each program

2.2.1 Significant classes within the Web application

As the PHP is procedural each function is treated as a class for the purpose of description.

index.php

Description of classes for the file index.php:

src="https://maps.googleapis.com/maps/api/js?key=&sensor=false">** This piece of javascript is part of the google maps API, it gathers a user key and allows access to the functionality of the google map.

function calcRoute() The calcRoute() function takes the parameters of an array, from there it will plot the path between all points in the array and direct the user to follow, drawing the path on the map.

function initialize() The initialize() function is the basis for booting the map. Within contains the while loop that accesses our MySQL database and returns all points for a given path. During this time it will display all points with their appropriate marker location and give the descriptions stored of each and every point.

$con Establishes connection to host based on server permissions.

$res Returns appropriate data from selected database for use with google map.

var LatLng, var ContentString These are variables concerning the retrieval of data from the database, and as such are unique per point. LatLng creates a new marker posted at the given points within our tables, the contentstring updates the information window with the stored point description.

google.maps.event.addListener This creates the information window given at every marker on the map. Creates a listener that waits for a mouse click to open the information window.

var marker = new google.maps.Marker Constructor for the marker, reads the position it should be set at, the animation accompanying it upon opening of the map, the drag mode, the icon and the map it should be assigned to.

google.maps.event.addDomListener(window, 'load', initialize) Creates map upon load of the window, runs the initialize function to ensure all points and paths are plotted upon creation. Opens the map within the specified window space.

upload.php: This page will receive the information as sent by the android client

2.2.2 Significant classes within the Android application

Basic descriptions of each significant class within the phone application. A more detailed description provided in the Interface description section.

Tour.java – stores information about each tour. This will include a string for the name, an array of locations, a string for the description, another string for the longer description.

HomeActivity.java – holds methods that take user input, iit is the home page of the application.

TourCreatorActivity.java – Creates the screen and GUI which takes user input to intern create a tour based on that information.

TourActivity.java – links going to adding and editing locations as well as stopping the tour and uploading it to the server.

LocationCreatorActivity.java – allows to put in the values for each location as well as the picture. This can then be save in the array of location or delete that location.

EditLocationActivity – Same activity as the LocationCreatorActivity except for instead of creating the location, it finds the location in the array list and changes its values.

ViewWalksActivity – shows all the walks stored in the array list and allows the user to edit it.

2.3 Modules shared between programs

The only modules that will be shared between the web application and the android side application is the JSON script that would pass information between the phone and the web storage page via http. The android would pack data into JSON regarding points and this would be sent to the web application to be parsed out and inserted into the relevant tables. The android user may specify as to whether a new path is being generated, and if this is true a new row must be added to the paths table to signify that this path and all points associated are separate entities. Similarly all points made under this new path would be associated on the database side via a pathID foreign key within the rows for each point. JSON must also be able to pack images into a string to be sent, which would allow for easy transmission and ensure that the data can be received and stored. Decoding of such data need only happen upon arrival at the database or on the page as and when requested.

2.4 Mapping from requirements to classes

This table shows each functional requirement that must be met, and which class provides its operation. This will ensure that all requirements are covered, and show the impact that changes to the design may have.

Functional requirement	**Class providing requirement**
FR1- Startup of software on Android device	HomeActivity.java
FR2 - Providing info about the whole walking tour	TourCreatorActivity.java
FR3 - Adding locations to the walking tour	TourActivity.java
FR4 - Adding photos to the walks	LocationCreatorActivity.java
FR5 - Cancelling walks	TourActivity.java
FR6 - Sending the walk to the server	TourActivity.java
FR7 - Switching from the WTC	All activity classes
FR8 - Trying out a created walking tour	index.php
FR9 - Saving data on server	database

Figure 1: Table showing the functional requirements as well as classes which provide them

3 DEPENDENCY DESCRIPTION

3.1 Component Diagram

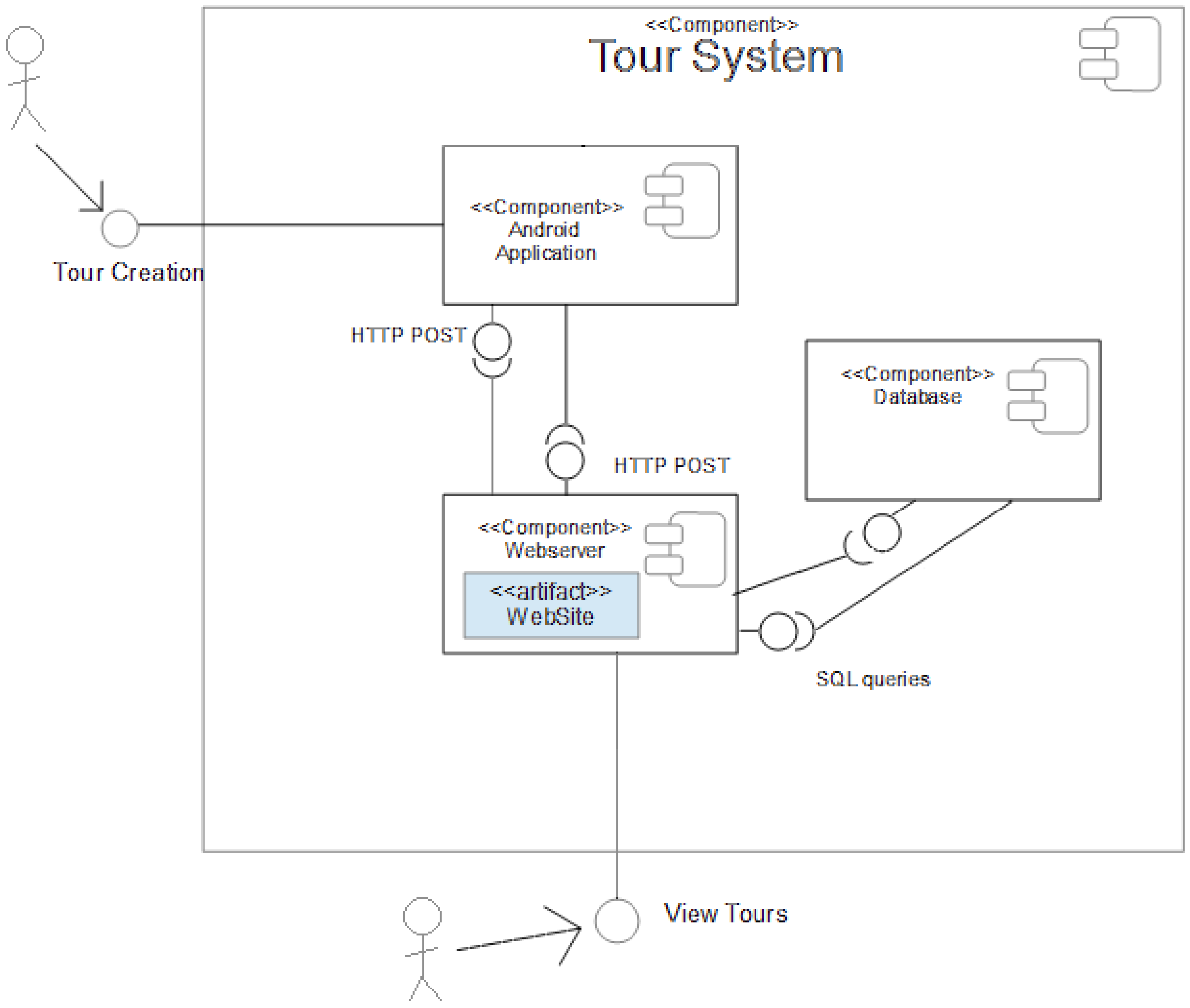

Figure 2: Component diagram

3.2 Inheritance Relationships

Activity classes will inherit from Android API packages.

4 INTERFACE DESCRIPTION

4.1 Web Class Interface

```
<DOCTYPE html>

<html>

/* Using HTML 5 Doctype to support our project. This ensures that the google
maps API will be fully supported on the latest web standard, that the phone
viewport can be resized and viewed for and there will be no implementation
issues concerning any new web tools we may wish to access or incorporate. */
<head>

<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />  <style
type="text/css">

html { height: 100% }

body { height: 100%; margin: 0; padding: 0 }

#map-canvas { height: 100% }

</style>

/* A temporary inline CSS for use with this test page. Only contains code
necessary for display of the map.*/

<script type="text/javascript"  src="https://maps.googleapis.com/maps/api/js?
key=****************************************&sensor=false">

</script>

/*Fetches a google maps API key to be able to use the google maps

functionality and tools across the site.*/

<script type="text/javascript">

var image = 'icon2.png';

var waypts[];

/*Variable name "image" accesses the location of the saved marker image, can be
removed to revert to default marker or file name can be changed to allocate new
marker should future developments occur. Variable name "waypts" is an array that
contains the lat and long values of points along the path to be taken for the
lines of direction to be drawn for the user. This can only hold a size of 8 or
the path will not draw due to API limits.*/

function calcRoute()

{

var request=

{

origin: new google.maps.LatLng(<?php=$start['lat]?>,<?php=$start['long']?>),

destination: new google.maps.LatLng(<?php=$end['lat]?>,<?php=$end['long']?>),

waypoints: waypts;

travelMode: google.maps.TravelMode.WALKING

};

<?php=$pointpath = mysql_query("SELECT * FROM pathlines WHERE `pathID` =
$_POST["path"] && `pointID` = $_POST["points"];
while($b=mysql_fetch_array($pointpath)
{>?
for(i=0;i<8;i++)
{
waypts.push[{<?php=$b['lat]?>,<?php=$b['long']?>}];
}
<?php=}?>
directionsService.route(request, function(response, status)
{
if (status == google.maps.DirectionsStatus.OK)
{
directionsDisplay.setDirections(response);
directionsDisplay.setOptions({suppressMarkers: true});
}
});
/*Calculates routes between two points. Uses an origin point to find start of
path, and destination point to use as end of path. Travel mode dictates what
type of route and paths are allowed. Walking indicates any available path over
land, driving may only use roads however. Walking is appropriate for prototype
path. The tool behind this to be used is google map's very own "polyline" type,
which will run through all of the points specified between two points selected.
Suppressed markers is a tool used to prevent the appearance of additional
markers dictating start/end points of a path and cluttering the GUI.*/
}
function initialize()
{
var mapOptions = {center: new google.maps.LatLng(52.413571,-4.073489), zoom:
14};
var map = new google.maps.Map(document.getElementById("map-canvas"),mapOptions);
var infowindow = new google.maps.InfoWindow();
window.directionsService = new google.maps.DirectionsService();
window.directionsDisplay = new google.maps.DirectionsRenderer();
directionsDisplay.setMap(map);
/*Initializes map for later, starts by creating variables and generates paths
within the window. Adds path to the map and sets the position at center when the
map is first displayed. Also sets the rate of zoom so the entirety of the path
is covered.*/
<?php
$con = mysql_connect("localhost","root","");
if (!$con)
{
```

```
die('Could not connect: ' . mysql_error());
}
mysql_select_db("pathdb", $con);
$res = mysql_query("SELECT * FROM points");
while($a = mysql_fetch_array($res))
{
?>
/*Creates a connection to my local hosting server, selects database to use and
then chooses the data from which tables I am currently interested in. Enters a
while loop to assign each marker and create it as such.*/
var LatLng = new google.maps.LatLng(<?=$a['lat']?>,<?=$a['long']?>);
var ContentString = "<b><?=$a['shortDesc']?></b></br><?=$a['longDesc']?>";
var marker = new google.maps.Marker(
{
map:map,
draggable:false,
animation: google.maps.Animation.DROP,
position: LatLng,
icon: image
});
marker.content = ContentString;
google.maps.event.addListener(marker, 'click', function(){
infowindow.setContent(this.content);
infowindow.open(this.getMap(),this);
});
/*Uses value (Lat,Long) to create point marker on map, assigns variable
"ContentString" which creates a bolded up title as the short description,
followed by a new line for the long description per point. Marker is generated
for our map, with drag disabled, a drop animation on generation, marker appears
at the given (Lat,Long) position and marker at point is assigned  to image
variable given earlier. Listener sets up the information to go into our
information window and sets the function to open the window upon the click of
the marker happening.*/
}
?>
calcRoute();
/*Closes the while loop and calls the path finding function between points.*/
}
google.maps.event.addDomListener(window, 'load', initialize);
/*Upon loading of page, the map is generated with it's appropriate window space,
and the function initialize is called and therein all paths are brought up and
all points are created.*/
```

```
</script>
</head>
<body>
<div id="map-canvas"/>
</body>
/*Creates the map on the website, closes all HTML concerned with page.*/ </html>
```

4.2 Android Interface Description

This section will provide an outline specification for each significant class within the Android system.

4.2.1 Tour.java

```
public class Tour implements Serializable {

private String name;
private String shortDescription;
private String longDescription;
private ArrayList<TourLocation> locations;
private ArrayList<Double> latitudes;
private ArrayList<Double> longitudes;

/**
* The constructor for a Tour.
* It has a name, a short Description and a Long Description.
* @param name The name of the Tour.
* @param shortDescription A short description that describes the route taken.
* @param longDescription A description that can be used to describe the route
* taken in more detail.
*/

public Tour(String name, String shortDescription, String longDescription) {
}


public void addLocation(TourLocation location) {
}

public void addWaypoint(double lat, double _long){
}

/**
* Attempts to convert a Tour into a Json string, so that it may be sent to the
database via HTTP Post.
* @return a Json String that holds the information of a single tour.
*/
public String toJSON() {
}

public String getName() {
}
```

```
public String getShortDescription() {
}

public String getLongDescription() {
}

public ArrayList<TourLocation> getLocations() {
}
```

4.2.2 LocationCreatorActivity.java

```
public class LocationCreatorActivity extends Activity implements
LocationListener {
private LocationManager locationManager;
private String provider;
private double latitude = 0;
private double longitude = 0;
private String imageFilePath;
private ImageView imageView;
private Tour tour;
private Bitmap savedImage;

protected void onCreate(Bundle savedInstanceState) {
}
public void onPhotoClick(View view) {
}
private void onSelectImage() {
}
 public void onCoordinateClick(View view) {
}
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
}
public void onLocationChanged(Location location) {
}
private void setupActionBar() {
}
public boolean onCreateOptionsMenu(Menu menu) {
}
public boolean onOptionsItemSelected(MenuItem item) {
}
public void onStartAddLocation(View view) {
}
```

```
public void onStartDeleteLocation(View view) {
}
protected void onResume() {
}
protected void onPause() {
}
public void onProviderDisabled(String arg0) {
}
 public void onProviderEnabled(String arg0) {
}
public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
}
```

4.2.3 HomeActivity.java

```
public class HomeActivity extends Activity {

protected void onCreate(Bundle savedInstanceState) {
}
public boolean onCreateOptionsMenu(Menu menu) {
}
public void onStartTourCreator(View view) {
}
public void onStartUpload(View view) {
}

public void onStartQuit(View view) {
}
}
```

4.4.4 TourCreatorActivity.java

```
public class TourCreatorActivity extends Activity {

protected void onCreate(Bundle savedInstanceState) {
}
private void setupActionBar() {
}
public boolean onCreateOptionsMenu(Menu menu) {
}
public boolean onOptionsItemSelected(MenuItem item) {
}


/**
* When Tour Creation begins, it stores the name, Short Description and Long
Description
* by creating a new instance of tour, passing in these parameters.
* If any field is left blank, a pop-up message is displayed, and the Tour * is
not saved.
* @param view the button that was clicked to call this method.
*/


public void onStartTour(View view) {
}
public void onStartTour(View view) {
}
}
```

4.4.5 TourActivity.java

```
public class TourActivity extends Activity implements LocationListener{
private static double TIME_BETWEEN_WAYPOINTS = 10000;
    private static boolean DEBUG = true;
    private LocationManager locationManager;
    private String provider;
    private double latitude = 0;
    private double longitude = 0;
    private double time = 0;
    private Tour tour;
protected void onCreate(Bundle savedInstanceState) {
}
public void onLocationChanged(Location location) {
}
public boolean onCreateOptionsMenu(Menu menu) {
}
public void onStartLocationCreator(View view) {
}
public void onStartDeleteTour(View view) {
}
public void onStartEditLocations(View view) {
}
public void onUpload(View view) {
}
protected void onResume() {
}
protected void onPause() {
}
public void onProviderDisabled(String arg0) {
}
public void onProviderEnabled(String arg0) {
}
public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
}
}
```

4.5.6 EditLocationActivity.java

```
public class EditLocationActivity extends Activity{
private ImageView imageView;
    private Tour tour =new Tour(null,null,null);
    private String imageFilePath;
    private double latitude = 0;
    private double longitude = 0;
    private ArrayList<TourLocation> locations;
    private int locationNumber;
protected void onCreate(Bundle savedInstanceState) {
}
public void onCoordinateClick(View view) {
}
public void onPhotoClickEdit(View view) {
}
private void onChangeImage() {
}
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
}
public void onStartEditLocation(View view){
}
}
```

4.5.7 ViewWalksActivity.java

```
public class ViewWalksActivity extends Activity {
    public int editLoc;
    ListView listView;
    private Tour tour =new Tour(null,null,null);
    private ArrayList<TourLocation>locations;
protected void onCreate(Bundle savedInstanceState) {
}
private void setupActionBar() {
}
public boolean onCreateOptionsMenu(Menu menu) {
}
public boolean onOptionsItemSelected(MenuItem item) {
}
}
```

4.5.8 JSON.java

```
public class JSON {
/**
*
* Prevents illegal characters from being sent in the Json Post.
* it adds a \ in front of illegal characters so that they do not
* get misinterpreted.
* @param string to be escaped for illegal characters.
* @return A corrected Json String, ready to be Posted.
*/
public static String quote(String string) {
}
```

4.5.9 Post.java

```
public class Post {
    private String urlString;
    private String postBody;
public Post(String urlString, String postBody) {
}
/**
* Attempts to Post a Json string via HTTP.
*
* @return a boolean value, to confirm whether the Post was
* successful
*/
```

```
public boolean send() {
}
/**
* Runs the send method, and if it fails,
* it attempts to save the tour locally on the device.
*/
public void sendAsync() {
}
```

4.9.10 Image.java

```
public class Image {
/**
* This method converts the Bitmap image taken from the camera for a location
* and converts it into a base64 String. This String will be send to the database
for storage
* via Json Post.
* @param image The image to be converted
* @return The base64 Converted image String
*/
public static String base64(Bitmap image){
}
```

4.9.11 FileManager.java

```
/**
* This class manages the local saving and loading of tours to the Android
* device.
* This is done so that in the event of a connection failure, the tours can be
* uploaded at a later
* date.
*/

public class FileManager {
/**
* This method serializes an array of tours to a .ser file. The file is stored
locally on the
* device running the application.
* If the file already exists on the device, then the file is ammended and the
new array of tours
* are saved with the old array of tours.
```

```
*
* @param tour The array of tours we wish to save locally.
*/
public void writeToFile(Tour[] tour) {
}
/**
* This method deserializes the tours stored in the file and turns it back into
an array of tours.
* It then removes the written file. The array of Tours is returned so that it
can then be converted
* into the JSON String to send to the Web-Server Database.
*
* @return The array of tours that has been deserialized from local storage.
*/
public Tour[] readFromFile() {
}
public void remove() {
}
/**
* This is the method used if an attempt to write on a file that already exists
takes place.
* In order to prevent the file from being overwritten, it is read back into an
array,
* and is combined with with the new array we are trying to save.
* The new, updated array is then written back into the file.
*
* @param tour This is the new array that must be combined with the original
array already stored in file, before being saved.
*/
public void append(Tour[] tour) {
}
```

4.9.12 Tour.java

```
public class Tour implements Serializable {
      private static String TOUR_JSON;
      private String name;
      private String shortDescription;
      private String longDescription;
      private ArrayList<TourLocation> locations;
      private ArrayList<Double> latitudes;
      private ArrayList<Double> longitudes;


/**
* The constructor for a Tour.
* It has a name, a short Description and a Long Description.
*
* @param name The name of the Tour.
* @param shortDescription A short description that describes the route taken.
* @param longDescription A description that can be used to describe the route
taken in more detail.
*/
public Tour(String name, String shortDescription, String longDescription) {
}
public void addLocation(TourLocation location) {
}
public void addWaypoint(double lat, double _long) {
}
/**
*Attempts to convert a Tour into a Json string, so that it may be sent to the
database via HTTP Post.
* @return a Json String that holds the information of a single tour.
*/
public String toJSON() {
}
public String getName() {
}
public String getShortDescription() {
}
public String getLongDescription() {
}
public ArrayList<TourLocation> getLocations() {
}
}
```

4.9.13 TourLocation.java

```
public class TourLocation {
      private static String LOCATION_JSON;
      private String name;
      private String description;
      private String image;
      private String base64Bitmap;
      private double latitude;
      private double longitude;
      private double time;
/**
* This is the constructor for a location in a Tour.
* @param name The name of a location. (E.g. "The Pug's tail")
* @param description The Descripion of a location. (E.g. "The local watering
whole for the crimnal puppy masterminds.")
* @param imagePath The image filepath of a picture taken at the location. (E.g.
" C:\Users\Benson\My Pictures\woof.bark")
* @param latitude The latitude coordinate of a location.
* @param longitude The longitude coordinate of a location.
* @param time The date/Time of when a location was recorded into the phone.
*/
public TourLocation(String name, String description, String imagePath, String
base64Bitmap, double latitude, double longitude, double time) {
}

/**
*Attempts to convert a location into a Json string, so that it may be sent to
the database via HTTP Post.
* @return a Json String that holds the information of a single location.
*/
public String toJSON() {
}
public String getName() {
}
public String getDescription() {
}
public String getImage() {
}
public double getLatitude() {
}
public double getLongitude() {
}
public double getTime() {
}
public void setName(String name) {
}
 public void setDescription(String description) {
}
public void setImage(String image) {
}
}
```

5 DETAILED DESIGN

5.1 Sequence diagram

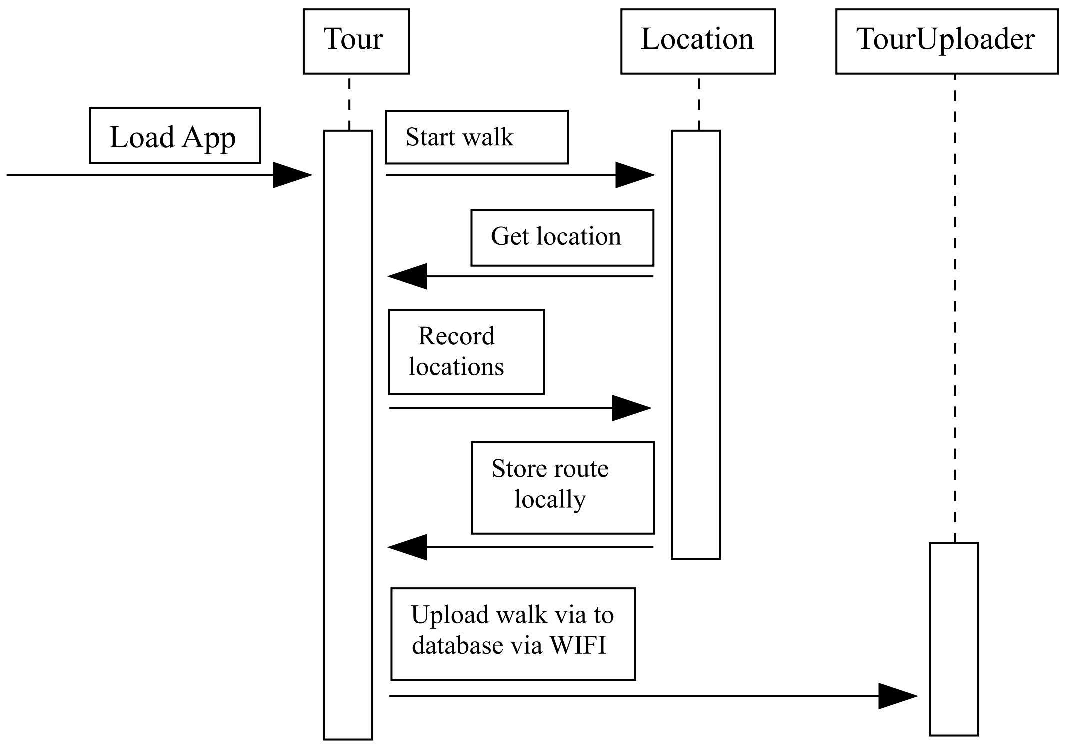

Figure 3: Sequence diagram

5.2 Significant algorithms

5.2.1 Significant algorithms for Web

The first significant algorithm to be concerned at for our backend of the map would be that of the marker positions, marker positions are projected using stored lat/long positions for points based on the path selected by the user.

$res = mysql_query("SELECT * FROM points where `pathID` = $path"); while($a = mysql_fetch_array($res)) {var marker = new google.maps.Marker}

This snippet of code outlines the selection process for markers on the map. At the top the markers are selected based upon the path they belong to, and then for every marker meeting this criteria, it is based onto the map based on it's latitude/longitude values along with the description of the point stored within the information window.

var flightPath = new google.maps.Polyline({path: flightPlanCoordinates,strokeColor: '#FF0000',strokeOpacity: 1.0,strokeWeight: 2});

The google maps API comes packed with a tool called "Polyline", this will allow the web page to sketch it's own paths around the map, in which the path of the Polyline will be drawn from an array of lat/long values under the "path" method of the Polyline class.

The methods for obtaining the array to give to the polyline involves a for loop that counts from 0 to 7 (8 values) and for each loop, will push the new lat/long value to the array.

Present on the page for the map will now be buttons that will cast POST calls to the same header page, altering the values used and will show different routes between points and different paths as and when called.

This is however the extent of the significant algorithms to be used in the maps functionality.

5.3 Significant data structures

Our significant data structure used is a schema made in JSON.

```
{
    "type": "object",
    "properies": {
        "name": {
            "type": "string",
            "description": "The name of the tour"
        },
        "short-description": {
            "type": "string",
            "description": "The short description of the tour"
        }
        "long-description": {
            "type": "string"
            "description": "The longer description of the tour"
        },
        "locations": {
            "type": "array",
            "description": "An array of all the locations in the tour",
            "items": {
                "type": "object",
                "description": "A single location",
                "properies": {
                    "name": {
                        "type": "string",
                        "description": "The name of the location to be displayed to the user"
                    },
                    "latitude": {
                        "type": "number",
                        "description": "Latitude in degrees"
                    },
                    "longitude": {
                        "type": "number",
                        "description": "Longitude in degrees"
                    },
                    "time": {
                        "type" "number",
                        "description": "Unix timestamp of the time the location was recorded"
                    },
                    "image": {
                        "type": "string",
                        "description": "Base64 encoded JPEG image"
                    }
                },
                "required": [
                    "name",
                    "latitude",
                    "longitude",
                    "time",
                    "image"
                 ]
            }
        }
    },
    "required": ["name", "short-description", "locations"]
}
```

This is the schema that we will be using. Essentially, Tours that we create in the android application are made in Java, which is converted into a Json post. This is done so the application must adhere to provide the correct data each time it sends information to the database for storage.

If the application tries to send a Post to the database, and the data sent does not conform to the JSON schema, it will fail to validate, and an error will be displayed.
The JSON Post can be interpreted in PHP due to the fact that it is language-independent. This means that we do not have to make any difficult conversions to the data that is sent back and forth between the Database and the Java application.

json_decode() can be used to convert JSON strings into PHP variables very easily, and this can then be stored in the database without much fuss. The function **json_encode()** can then be used to send information back to the Handheld device.

5.4 Entity Relationship Diagram

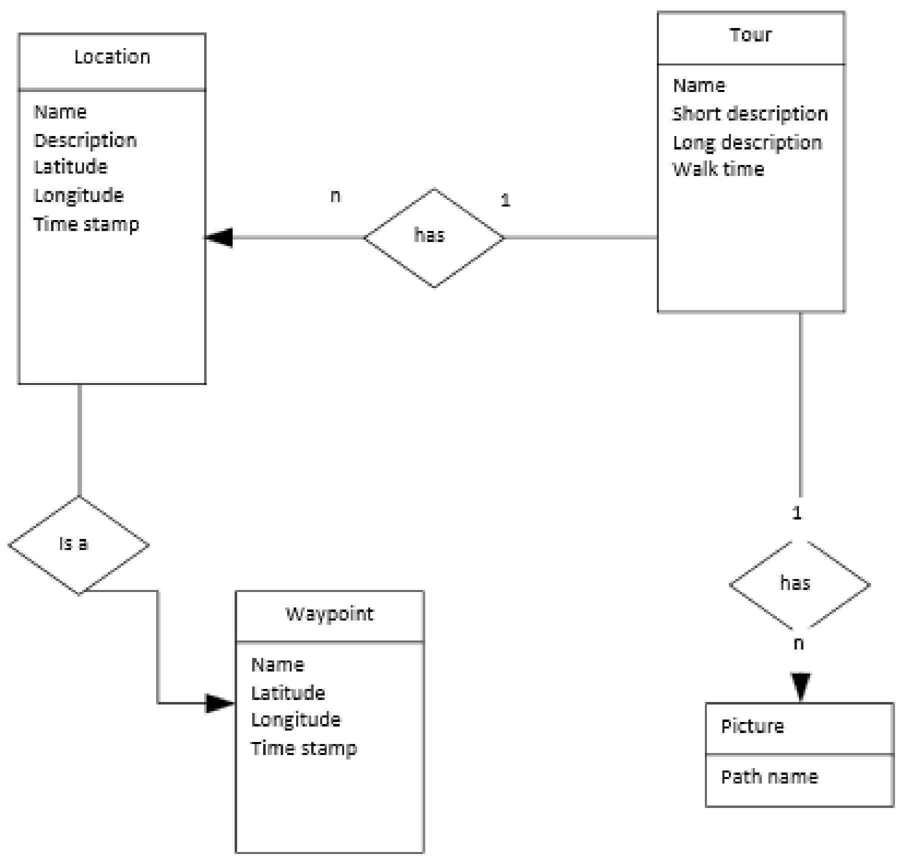

Figure 4: Entity relationship diagram

5.5 Class Diagram

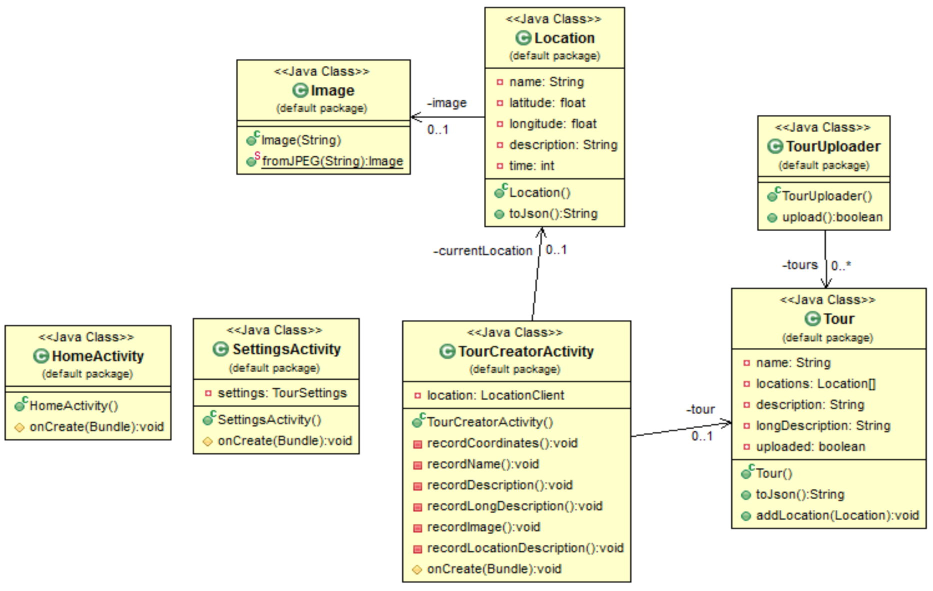

Figure 5: Class diagram

The above class diagram has changed since finishing the whole of the Android code.

6 REFERENCES

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

APPENDICES

Document History

Version	CCF No.	Date	Changes made to the document	Changed by
0.1	N/A	25/11/13	Initial creation.	LPS1
0.2	N/A	01/12/13	Introduction added	LPS1
0.3	N/A	01/12/13	Mapping requirements to classes section added	LPS1
0.4	N/A	04/12/13	Added significant classes for web,	LPS1

			programs in system and decomposition description	
0.5	N/A	04/12/13	Added Significant classes for Android	LPS1
0.6	N/A	05/12/13	Interface descriptions for Android added, squence diagram added, significant data structures added, added algorithms for web.	LPS1
0.7	N/A	06/12/13	Document revised & vetted.	LPS1
1	N/A	28/01/14	Major formatting issues corrected.	LPS1.
1.1	N/A	17/02/14	Revised and updated the document. Each section updated based on the requirements.	FIK