Software Engineering Group Project
Design Specification

Author:	Christopher Edwards;Douglas Gardner;Luke Horwood;Jostein Kristiansen;Ben Rainbow;Ashley Smith;James Woodside;Dillon Cuffe
Config Ref:	SE_02_DS_001
Date:	2014-01-29
Version:	0.4
Status:	Released v1.1

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 whole design specification aspect of the Android Application that is going to be created. Within this document there will be outlines for the design of the significant classes and detailed mapping of the requirements on the classes. As well as the defining dependency descriptions, manufacturing the interface description and having a detailed design for the entire system.

1.2 Scope

The design specification shows all the different components that will need to be implemented and will describe how components like the interface will work. In doing this it will allow for the group to access information and work in conjunction with the Requirements Specification.
All information on this document will need to be read and reviewed by all members of the group.

1.3 Objectives

The objectives of this document are to outline and define clearly the entire layout and design of the Walking Tour Android Application. The requirements of this document are set by the customer's standards and this document will take into consideration those points and address them by creating an application. This will clearly address and implement those requirements of the customer.
Below will be the complete design of the application in conjunction to these requirements, examples of this will be found in the form of how the software is structured, components of the software and how the interface works and reacts to user input.

2. DECOMPOSITION DESCRIPTION

2.1 Programs in system

The Walking Tours application will consist of two main components. The two applications are the Android application and the web processing application. The web application is dependent on the Android application, and in some sense the Android application depends on parts of the web application.

The Android application:

The Android application will be the main application in this project. The Android application will consist of several activities and classes, which will be used in conjunction with user interactions and Google API's to record a user's walk.

This application revolves around the process of initially creating a walk with a given name and descriptions. This information is then stored within the application (in an easily accessible location) and the users current GPS location is retrieved. This GPS location can then be used to provide the user with a localised map of the surrounding area, which can then be used to display the users movements (requires the ability to constantly record GPS location) and also allow for the addition of flagging up "Points of Interest" along the current route.

These "Points of Interest" will also prompt the user for a name, description and the option of taking a photo/selecting an image to be associated with said "Point of Interest". These "Points of Interest" will also be displayed on the current map and their location on this map will be determined by the GPS location where the

user creates the point of interest. At this point the coordinates of the user's location will be assigned to the "Point of Interest" which allows for placement upon the map.

Once the user is satisfied with the current walks situation (e.g. all "Points of Interest" added and walk finished) then they can upload the results to the server. The user will be prompted to make sure they do in fact want to upload the walk and unwanted clicks can happen accidentally on touch screen devices.

Uploading to the server will require a device (running Android) with network connectivity of some variation in order to access the server, the information will be sent to the server as a HTTP POST request. This is where the application is reliant on the web application, if the server is unavailable then the application cannot meet the functional requirements (FR 6).

This part of the application will run on any Android enabled device running Android version 2.3+ (Gingerbread onwards) and a minimum API level of 10.

The web application:

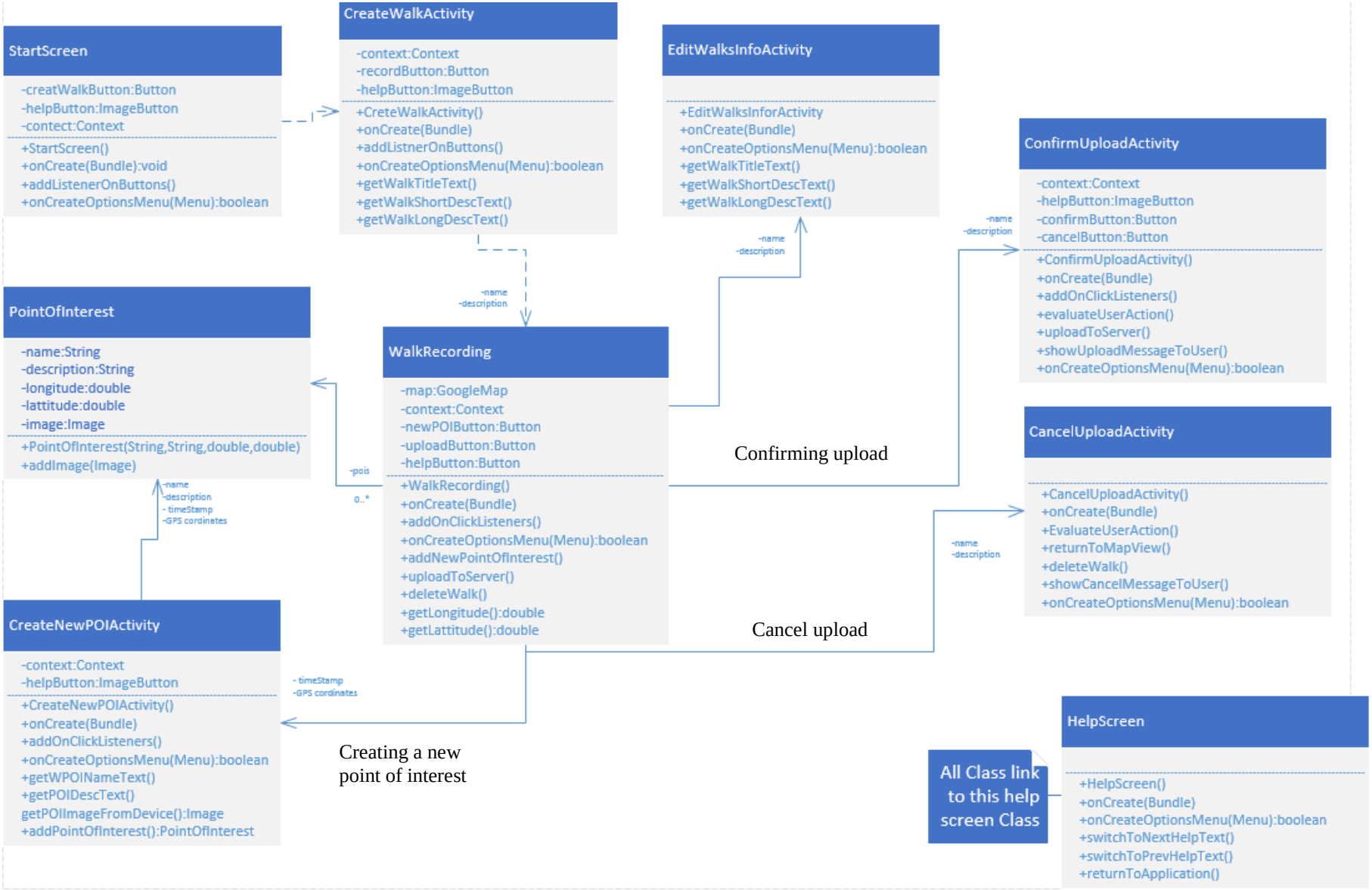
The web application refers to the website and the processing server required in order, producing a fully functional product.

On its own, the website will be able to retrieve information from a database; this information will be directly related to the number/details of currently saved walks sent from the android application. The information retrieved from the database will then be displayed initially as a list of all the stored walks (listed by title). Then a visitor to the website can click a walk and be shown the walks route, the information, the walks duration and all points of interest associated with the route.

The web application is heavily dependent on the Android application. If no information (e.g. walks) is being sent to the server from the application then the database will contain little or no entries resulting to none or few walks being displayed on the website. The main dependency is the web application with the Android application, as saved walks are needed to be on the web application to produce effective results.

This part of the application will be on the web server and any modern up to date browser.

2.2 Significant Classes:



StartScreenActivity (FR 1, FR 7):

The FrmHome class represents the initial screen presented to the user. This screen will be essential in the running of the application, as it houses a button which calls for the creation of the walk; until this button is clicked the user will not progress from this initial state, where only the FrmHome screen is displayed. In order to achieve this there will be a simple onClick listener attached to the button, which will simply call a different activity to be brought into focus, this activity will be CreateWalkActivity.

CreateWalkActivity (FR 2, FR 7):

The CreateWalkActivity class represents the screen where the user will be prompted for the information related to the walk. This information will include the walks name, short description and its longer more detailed description. This class also calls the creation of the WalkRecording class which will be the map/route display to the user.

WalkRecording (FR3, FR 4, FR 5, FR 6, FR 7, FR 9):

The WalkRecording class is used to model the current route using graphical representation. The map will display the user's current position and the current time elapsed along the route, as well as displaying all (if any) current Points of Interests along the current walk's path. The user can also create new Points of Interest and add them to the walk when viewing this screen.

CreateNewPOIActivity(FR 3, FR 4, FR 7):

The CreateNewPOIActivity class allows the user to specify the information required in order to create a new Point of Interest which can then be assigned to the current walk and stored locally ready for the server upload. This class is vital if the application is to achieve maximum functionality.

PointOfInterest (FR 3, FR4):

The PointOfInterest class is used to specify/hold the information related to each Point of Interest. The user can create multiple Points of Interests during one walk recording. It is essential to modulate the system, having a separate class to store/model the Points of Interest to allow this functionality.

CancelUploadActivity (FR 5, FR 7):

This class provides the user with the ability to cancel the upload to the server. This prevents the user accidentally uploading a walk that is unfinished or unwanted. The user will be prompted to make sure they understand that the walk will not be uploaded.

ConfirmUploadActivity (FR 6, FR 7):

This class provides the user with the ability to upload the walk to server. This allows the user to view the walk they have just recorded/uploaded via the web application.

Main functions of the web application:

Understanding HTTP POST requests (FR 6):

The server will receive a POST request from the android application. It will parse this POST request and send the newly formatted information to the database system, which can then store the information in the correct tables in the database, ready for viewing on the website.

Storing information in the database (FR 8, FR 9):

There will be a DBMS (Database management system) in place to provide a suitable environment to externally store the information generated during the user's interaction with the Android application. This database can they be parsed by a script to produce the intended results when a person visits the website.

The website (FR 8, FR 9):

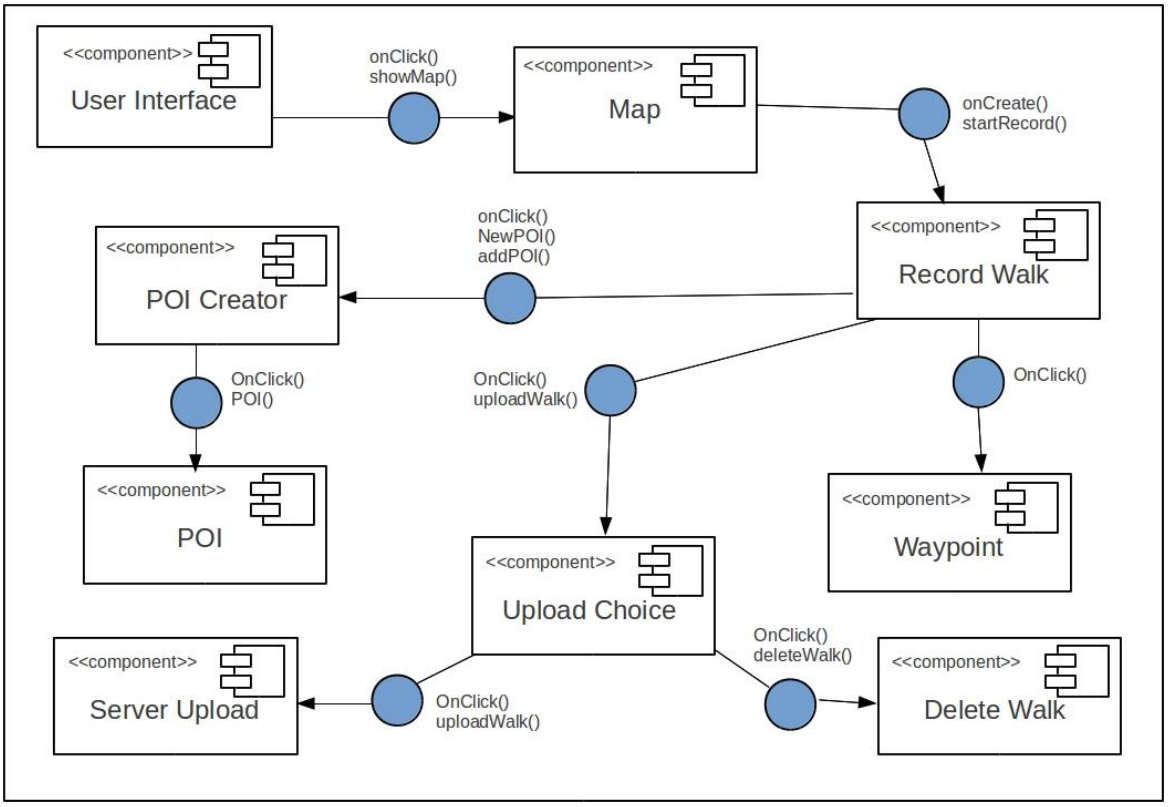
The web application will include a series of web pages which provide a suitable environment for the display of the stored walks information. This website will allow for user interaction and the user will be able to select which walk they want to view, this will directly reflect what information is retrieved from the database, thus producing different outputs for each walk.

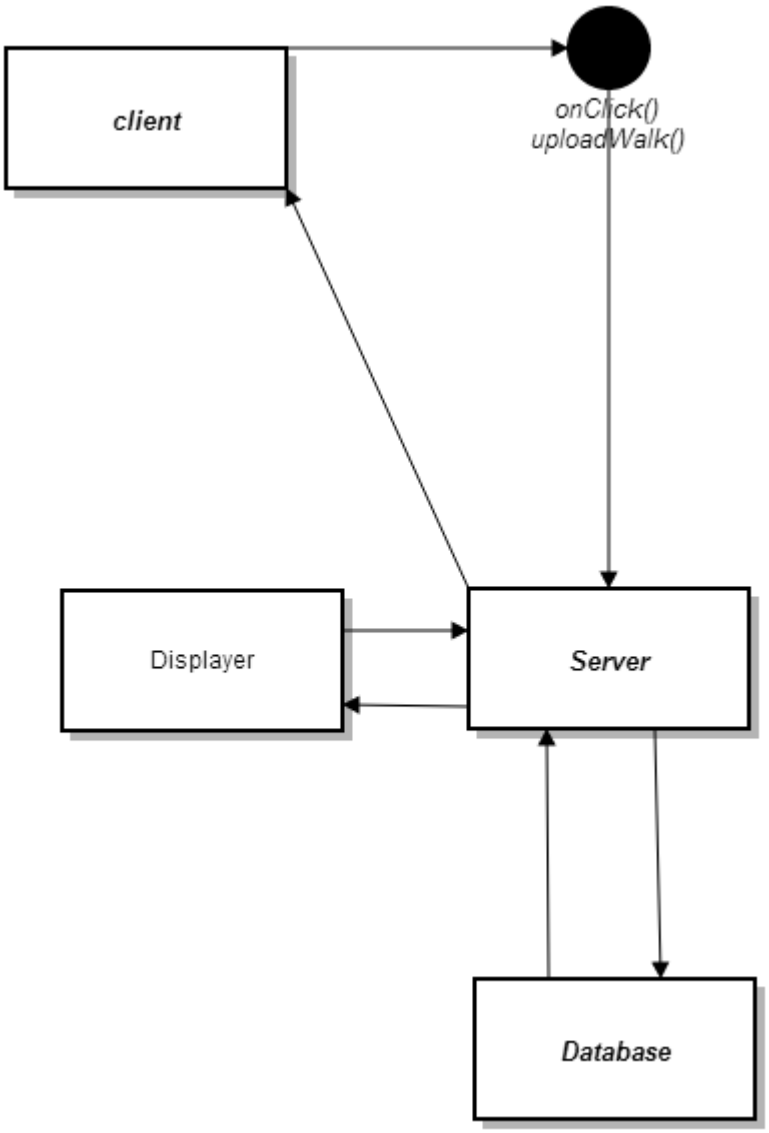
2.2 Mapping Function Requirements

Functional Requirement:	Classes providing requirement:
FR 1	StartScreenActivity
FR 2	CreateWalkActivity
FR 3	WalkRecording, CreateNewPOIActivity, PointOfInterest
FR 4	WalkRecording, CreateNewPOIActivity, PointOfInterest
FR 5	WalkRecording, CancelUploadActivity
FR 6	WalkRecording, ConfirmUploadActivity, Understanding HTTP POST requests (Server)
FR 7	StartScreenActivity, CreateWalkActivity, CreateNewPOIActivity, CancelUploadActivity, ConfirmUploadActivity
FR 8	Storing information in the database, the website
FR 9	WalkRecording, Storing information in the database, the website

3. DEPENDENCY DESCRIPTION

3.1 Component Diagram





How the classes map to the components
The component diagram shows the main system component which mainly relates to the Android application and how communication across the several modules and the server is accomplished. The below outlines the main ideas and principles behind this internal communication procedure:

User Interface:

The majority of the Android Classes will represent the user interface, as most of them extend Activity, thus providing a 'in focus' task on screen for the user to interact with. The only class which does not provide the user with some form of user interface directly is the PointOfInterest class. This is because it is the fundamental layout/data structure for every PointOfInterest (Location) on the walk, so the application user does not need to know how this information is stored or maintained.

The server also plays a role in the user interface as it provides the ability to store the walks in a database, which can then be accessed using MySQL queries. The walks can then be displayed on a webpage(s) in order to provide the user interface element of the web application.

Displaying the map component:

The map component is split between the two applications; both the Android client and the web application have functional requirements that specify a map must be displayed.

In regards to the Android application the main classes which feature in the displaying of the map and locations upon said map are as follows;

The WalkRecording class relates to the activity in which the actual map will be visible to the user. This will use the GoogleMaps for Android API's in order to achieve this. This is not the only class which will handle the displaying of the map for the Android client as it is a functional requirement that you can add locations to this map. This will be done using the interfaces provided in CreateNewPOIActivity class and the data stored in PointOfInterest class. The information will be relayed to the WalkRecording activity using the ability to add extra information to intents as well as using the provided startActivityForResult method, provided as part of the Android programming language specification.

The web application will display a map on the webpage and update the locations on this map based on the users selected walk. Once again the displaying of the map will be achieved using the GoogleMap's API and potentially JavaScript and PHP.

Recording a walk (GPS)

As an extension to the preceding paragraph which is related to the displaying of the GoogleMap inside the Android client, the requirement to record the walks GPS coordinates will also be handled using the WalkRecording class. This class will at some point use the LocationManager API's in order to access the network state of the device in order to achieve an accurate list of GPS coordinates along the walk. These coordinates can then be used to "trace" the route in which the walk has taken place.

POI creator/ Creating new PointsOfInterest/ POI

The ability to create new PointsOfInterest (locations) for the walk is handled using the interface provided in the CreateNewPOIActivity class. This class will provide a simple to use interface which will be used to enter information related to a new PointOfInterest. The information entered will be stored in respective variables/data structures in the POI class, and feedback to the WalkRecording activity which, as previously mentioned will handle where this POI will be displayed on the map object.

Upload choice/ Upload to server

The user must be prompted when upload to make sure that they do in fact want to upload to the server. The Android application will prompt the user using a User Interactive Dialog inside the ConfirmUploadActivity class. If the user does in fact mean to upload the walk to the server, then this will also be handled in the same class using some variation of a new thread or ASyncTask, in order to prevent possible complications associated with performing network connections on the main thread in Android development.

Deleting a walk

If the user does not want to upload the walk then the CancelUploadActivity will handle the cancellation of the walk upload. It states that if a walk does not want to be uploaded, then it must be deleted and a suitable prompt and warnings should be displayed to the user. If the walk is to be deleted then the CancelUploadActivity class will call the correct deletion methods in the WalkRecording class, this will effectively reset the application removing all points of interest and clearing the current map object of all information.

3.2 Inheritance relationships

There are two main inheritance relationships in the Android side of the walking tours application. These relationships are required in order to interact with the device and the provided API's as well as displaying information to the user in a valid format.

Activity (Android provided superclass):
The main inheritance relationship is the link between the Android Activity class and several classes in the application. The Activity superclass allows your designed screen to become a fully interactive screen when viewed on the android device, this is required to switch between views (e.g. map to upload screen). All of the classes that are listed against FR7, will extend Activity.

FragmentActivity (Android provided superclass):
FragmentActivity is the Class provided by Google Android API's. This Fragment Activity enables the application to display an instance of a GoogleMap object to the user. This is part of the functional requirements

as without this inheritance you cannot effectively deploy a solution using GoogleMaps effectively. This class is inherited by WalkRecording.

4. INTERFACE DESCRIPTION

4.1 Class Interface for CancelUploadActivity

```
//this class displays the cancel upload screen, ensuring the user wants to cancel the current walk

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

//outline for CancelUploadActivity class

public class CancelUploadActivity extends Activity {
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Creates an activity using the Activity super class
        *  The screen is then set to display the "cancel upload activity" interface.
        */
    }

    public void EvaluateUserAction() {
 //This method verifies that the user wants to cancel the upload.
    }

    public void returnToMapView() {
   //This takes the user back to the screen that has the map of the current walk displayed.
    }

    public void deleteWalk() {
  // This method removes all data for the walk that has been created so far.
    }
```

```
    public void showCancelMessageToUser() {
    //A message is displayed to the user telling them that the walk they created has been cancelled
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
        getMenuInflater().inflate(R.menu.cancel_upload_activty, menu);
        return true;
    }
}
```

4.2 Class Interface for ConfirmUploadActivity

```
//this class displays the confirm upload screen, ensuring the user wants to upload the current walk to the server.
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageButton;

public class ConfirmUploadActivity extends Activity {
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

    private Context context;
    private ImageButton helpButton;
    private Button confirmButton, cancelButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Creates an activity using the Activity super class
        *  The screen is then set to display the “confirm upload activity” interface.
        *  A help button will also be added using the addOnClickListeners() method so that the user can get
        *  help with anything they need in regards to the app.
        */

    }
```

```
    public void addOnClickListeners(){

        //adds click functionality to the help button which takes the user to the help screen (HelpScreen.class)



    }

    public void EvaluateUserAction() {
        //checks to see if the user really wants to proceed with the upload

    }

    public void uploadToServer() {
        //Sends all data regarding the walk that has been created to the server

    }

    public void showUploadMessageToUser() {
       /* A message is displayed to the user telling them that the walk they created has been successfully
       *  uploaded to the server.
       */



    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
        getMenuInflater().inflate(R.menu.confrim_upload, menu);
        return true;
    }

}
```

4.3 Class Interface for CreateNewPOIActivity

//this class displays the create new point of activity screen, allowing the user to create a new point of interest and set the name, description and image.

```
import android.media.Image;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;

public class CreateNewPOIActivity extends Activity {
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

    private Context context;
    private ImageButton helpButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Creates an activity using the Activity super class
        *  The screen is then set to display the "create new poi activity" interface.
        *  A help button will also be added using the addOnClickListeners() method so that the user can get
        *  help with anything they need in regards to the app.
        */
    }


    public void addOnClickListeners() {


       //adds click functionality to the help button which takes the user to the help screen (HelpScreen.class)


    }



    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
        getMenuInflater().inflate(R.menu.confrim_upload, menu);
        return true;
```

```
    }

    public String getWPOINameText() {

 //the name of the point of interest is returned in the form of a String.
        return "DEFAULT VALUE";
    }

    public String getPOIDescText() {

  //the description is returned in the form of a String.
        return "DEFAULT VALUE";
    }

    public Image getPOIImageFromDevice() {

   //the image is returned in the form of an Image.
        return null;
    }

    public PointOfInterest addPointOfInterest(){

  //adds this PointOfInterest to the walk
    }

}
```

4.4 Class Interface for CreateWalkActivity

//this class displays the create walk screen, allowing the user to create a new walk and give it a title, short description and long description.

```
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageButton;

public class CreateWalkActivity extends Activity {
```

```
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

    private Context context;

    private Button recordButton;

    private ImageButton helpButton;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Creates an activity using the Activity super class
        *  The screen is then set to display the "create walk activity" interface.
        *  A help button will also be added using the addOnClickListeners() method so that the user can get
        *  help with anything they need in regards to the app.
        *  A record button will also be added using the addOnClickListeners() method that takes the user
        *  to the Walk Recording screen which monitors the walk as it progresses.
        */

    }

    public void addListenerOnButtons(){

        /*  adds click functionality to the help and record buttons which take the user to the help
        *  screen (HelpScreen class) and record screen (WalkingRecording class) respectively.
        */

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
        getMenuInflater().inflate(R.menu.create_walk, menu);
        return true;
    }



    public String getWalkTitleText() {

// The name of the walk is returned in the form of a String.
```

```
            return "DEFAULT VALUE";
        }



        public String getWalkShortDescText() {

       // the short description is returned in the form of a String.
            return "DEFAULT VALUE";
        }

        public String getWalkLongDescext() {

        // the long description is returned in the form of a String.
            return "DEFAULT VALUE";
        }
}
```

4.5 Class Interface for EditWalksInfoActivity

```
//this class displays the edit walk screen, allowing the user to modify the title, short description and long description for the current walk.
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class EditWalksInfoActivity extends Activity {
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            /* Creates an activity using the Activity super class
            *  The screen is then set to display the “edit walk activity” interface.
            */
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
            getMenuInflater().inflate(R.menu.edit_walks_info, menu);
            return true;
        }
```

```
    public String getWalkTitleText() {
  // the walk title is returned in the form of a String.
        return "DEFAULT VALUE";
    }

    public String getWalkShortDescText() {
 //the short description of the walk is returned in the form of a String.
        return "DEFAULT VALUE";
    }

    public String getWalkLongDescext() {
   // the long description of the walk is returned in the form of a String.
        return "DEFAULT VALUE";
    }

}
```

4.6 Class Interface for HelpScreen

```
// this class displays the help screen, allowing the user to cycle through different tips.
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class HelpScreen extends Activity {
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Creates an activity using the Activity super class
        *  The screen is then set to display the “help screen” interface.
        */
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
         // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
         getMenuInflater().inflate(R.menu.help_screen, menu);
         return true;
    }
```

```
    public void switchToNextHelpText() {
  //changes the help displayed on the screen to the next help tip/advice.

    }

    public void switchToPrevHelpText() {
   // changes the help displayed on the screen to the previous help tip/advice.

    }


    public void returnToApplication() {
        // the user is taken back to the page that they were last on before they accessed the help page.

    }

}
```

4.7 Class Interface for PointOfInterest

// this class allows for the creation of a 'PointOfInterest' object where each object has a name, description and values for both longitude and latitude. Some of these objects also have an image.

```
import android.media.Image;

public class PointOfInterest {


    private String name, description;
    private double longitude, latitude;
    private Image image;

    public PointOfInterest(String name, String description, double longitude, double latitude) {

   // assigns the values passed in to the method as parameters of an individual point of interest.
        this.name = name;
        this.description = description;
        this.longitude = longitude;
        this.latitude = latitude;
```

```
    }

    public void addImage(Image im){
   // sets the value of the image to the image passed in to the method.
        this.image = im;
    }


}
```

4.8 Class Interface for StartScreen

```
// this class displays the start screen, allowing the user to either create a walk or view the help screen
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.Toast;

public class StartScreen extends Activity {
// Activity is extended in order to implement and manage multiple screens with varying methods/activities.

    private Button createWalkButton;
    private ImageButton helpButton;
    private Context context;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Creates an activity using the Activity super class
        *  The screen is then set to display the "start screen" interface.
        *  A help button will also be added using the addOnClickListeners() method so that the user can get
        *  help with anything they need in regards to the app.
        *  A create walk button will also be added using the addOnClickListeners() method that takes the
        *  user to the CreateWalkActivity screen, allowing the user to create a new walk.
        */
```

```
    }

    public void addListenerOnButtons() {
        /*adds click functionality to the help and create walk buttons which take the user to the help
        *screen (HelpScreen class) and create walk screen (CreateWalkActivity class) respectively.
        */
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
        getMenuInflater().inflate(R.menu.start_screen, menu);
        return true;
    }

}
```

4.9 Class Interface for WalkRecording

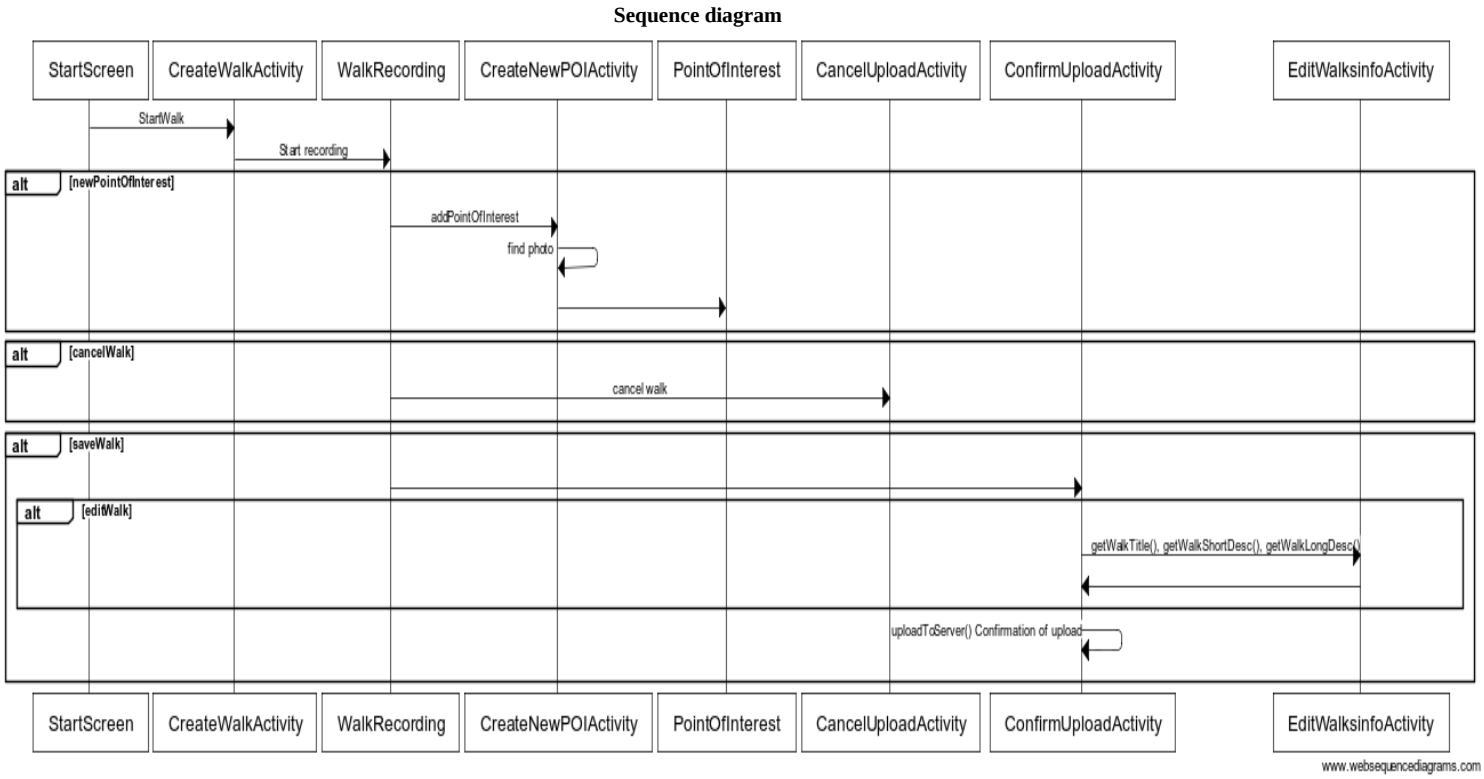
```
// this class displays the record walk screen and allows the user to add new points of interest, access the help
screen and view the current map.
import java.util.Vector;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.SupportMapFragment;

import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageButton;

public class WalkRecording extends FragmentActivity {
// FragmentActivity is extended in order to implement the map where the walk will be represented.
```

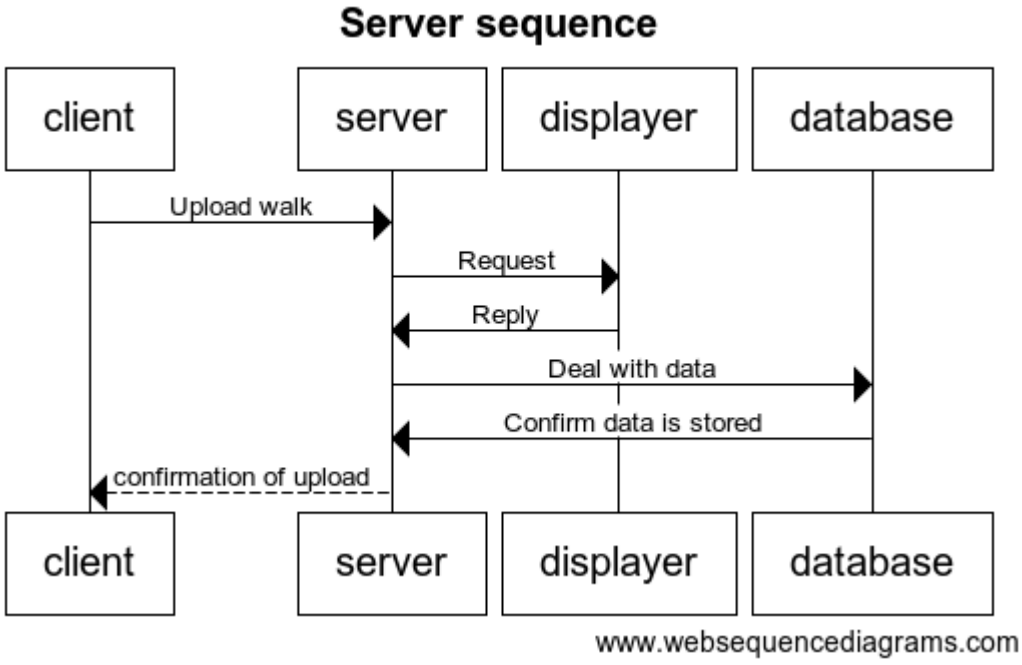
```
private GoogleMap map;
private Context context;
private Button newPOIButton, uploadButton;
private ImageButton helpButton;
private Vector<PointOfInterest> pois;

@Override
protected void onCreate(Bundle savedInstanceState) {
    /* Creates an activity using the Activity super class
    *  The screen is then set to display the "create walk activity" interface.
    *  The map is added to the screen using the FragmentActivity super class
    *  A vector of PointOfInterest is created in order to allow for numerous points of interest allowing
    *  them to be easily managed.
    *  A help button will also be added using the addOnClickListeners() method so that the user can get
    *  help with anything they need in regards to the app.
    *  A new point of interest button will also be added using the addOnClickListeners() method that
    *  takes the user to the CreateNewPOIActivity screen, allowing the user to create a new walk.
    * An upload button is added, allowing the user to finalise the walk and have it uploaded to
    * the server by going to the ConfirmUploadActivity screen.
    */

    }

@Override
public boolean onCreateOptionsMenu(Menu menu) {
     // Inflate the menu; this adds items to the action bar if it is present based on the current interface.
     getMenuInflater().inflate(R.menu.walk_recording, menu);
     return true;
}

public void addNewPointOfInterest() {
// adds a point of interest to the vector pois.

}

public void uploadToserver() {
// Sends all data regarding the walk that has been created to the server

}
```

```
        public void deleteWalk() {
     // removes all data that has been created so far and stops creating the walk



        }

        public double getLongitude() {
      // the longitude is returned to the app in the form of a double
              return 0;
        }

        public double getLatitude() {
    // the latitude is returned to the app in the form of a double
              return 0;

        }

}
```

5. DETAILED DESIGN

5.1 Sequence Diagram





5.2 Walking Tour Displayer

5.3 Overview

The Walking Tour Displayer is the second half of the product that we have been tasked to create. It is a web application that is used in conjunction with the Walking Tour Creator. The purpose of the Walking Tour Displayer is to show the user the routes they have saved via the Walking Tour Creator to a database.

5.4 Database

The Walking Tour Displayer requires a database, populated with routes created by the Walking Tour Creator. The Walking Tour Displayer will also accept uploads from the Walking Tour Creator and will store such uploads in the database accordingly.

5.4.1 Technologies

The database must be one utilising SQL, as specified in Functional Requirement 9. As such, the developers of the product have decided to use MySQL, or its close fork, MariaDB.

5.4.2 Structure

The structure of the database has been defined in the appendix to the Requirements Specification (SE.QA.RS).

This document uses SE.QA.RS version 1.4. Please note: more recent versions of the Requirements Specification may alter the requirements significantly, and have not been taken into account for this iteration of the document.

Design Constraint 3 makes it clear that the structure is mandated and cannot be modified from the specification.

The database has four tables: a “walks” table, a “locations” table, a “places” table, and a “photo” table.

5.4.2.1 List of Walks

The first table stores basic information about each individual route. The word ‘route’ is used interchangeably with ‘walk’. The routes table, which shall be named **tbl_routes**, contains seven fields, thus:

Field name		Field type	(SQL type)	Description / notes
id	PK	Integer	int	Primary key.
Title		Text	varchar(255)	Name of the walk
shortDesc		Text	varchar(255)	Subtitle of walk
longDesc		Text	varchar(1024)	Longer description
hours		Float	float	Hours a walk should take
distance		Float	float	Kilometres a walk takes

5.4.2.2 Location

The second table, **tbl_locations**, stores details of Points of Interest displayed on each walk.

Field name		Field type	(SQL type)	Description / notes
id	PK	Integer	int	Primary key.
walkId	FK	Integer	int	Foreign key from tbl_routes
latitude		Float	float	A decimal representation of GPS coordinates, using WGS84.
longitude		Float	float	
timestamp		Float (?!)	float	Seconds elapsed since walk began

5.4.2.3 Place Description

The third table, **tbl_places**, is used to associate locations with routes. The Requirements Specification states that the ID is used to order items within a walk.

Field name		Field type	(SQL type)	Description / notes
id	PK	Integer	int	Primary key.

locationId	FK	Integer	int	Foreign key from tbl_locations
description		Text	varchar(255)	Description of location

5.4.2.4 Photo Usage

The final table, **tbl_images**, is used to associate pictures with places.

Field name		Field type	(SQL type)	Description / notes
id	PK	Integer	int	Primary key.
placeId	FK	Integer	int	Referenced from tbl_places
photoName		Text	varchar(255)	JPEG filename (without extension)

5.4.3 Web Application

The application shall be a PHP application running on an Apache HTTP server. Both PHP and Apache HTTP Server are tried, tested, and trusted technologies. To allow quick development, the webpages will use the Bootstrap front-end framework for a springboard for HTML and CSS templates.

5.4.3.1 Requirements

The requirements for the Web Application, also known as the Walking Tour Displayer, can be found in the Requirements Specification (SE.QA.RS).

The relevant Functional Requirements are Functional Requirements 6, 8 and 9. For convenience, FR8 is replicated below:

> There will be a separate program (web application), the Walking Tour Displayer (WTD) that lets the user select a walk to display. When a walk is selected, the user will see each of the places included in the walk displayed at the correct coordinates on a map. The user will be able to choose a place, and see the correct details for that place within the selected walk, including textural [sic] descriptions and images.

The requirements for the Walking Tour Displayer seem to be as follows:

- There is a web application known as the Walking Tour Displayer
- It shall have a list of all routes generated by users, which will show the name and sub-title of each
- A user can select a route and view a second page with more detailed information to do with the route
- This Route View shall have a longer description of the route
- The Route View will have a map of the route, with all locations along the way shown
- A user can click a location and view images and descriptions associated with such a route

5.4.3.2 File Structure

Although, theoretically, the entire web application could be written in one long index.php file, it is not good practice. The scripts will be split into a couple of directories as follows:

- The root directory will contain the entry-points for the server, for example, index.php and upload.php
- An includes directory will contain functions and code used in multiple scripts. These functions will be included through use of the require and include functions built into PHP.
- An uploads directory will contain any JPG files the Walking Tour Creator has uploaded to the server.
- A resources directory will contain static assets, such as CSS and JavaScript.
- A templates directory will contain templates used to define the layout of the rendered HTML pages.

5.4.3.3 Viewing Walks

The initial homepage index.php will show a list of all walks uploaded to the server, displaying in a table the names and subtitles of each. Clicking on a name of a walk will take you to a second page (e.g. index.php?id=2) that will show a map of the route. The map shall show the points of interest specified in the walk, and interacting with a point of interest will display any photographs or descriptions associated with it.

5.4.3.4 Uploading Walks

FR6, replicated below, defines the way that the server communicates with the client.

> The user should be able to save the walk, by sending it to a server where it is saved into a database. The message should be formatted as a Multipurpose Internet Mail Extensions (MIME) message and sent to the server via an HTTP POST to a predefined URL.

However, FR6 gives considerable leeway in the way that the data is encoded. It is proposed that the client will send a JSON ('JavaScript Object Notation') formatted request to the server (at upload.php), encoding the data that the server should save into the database. For example:

```
{
    "authorization": {
        "hash": "76EADA9B070BB27659359220A460C264C34745A9",
        "salt": "swordfish"
    },
    "walk": {
        "title": "Whitehall Wander",
        "shortDesc": "A short walk around Westminster",
        "longDesc": "A walk around London, viewing sights such as Downing
Street, Trafalgar Square, and Scotland Yard",
        "locations": [
            {
                "latitude": 51.503396,
                "longitude": 0.127640,
                "timestamp": 0,
                "descriptions": [
                    "10 Downing Street"
                ],
                "images": [
                    "no10door",
                    "primeminister"
                ]
            },
            {
                "latitude": 51.506758,
                "longitude": 0.128692,
                "timestamp": 20,
                "descriptions": [
                    "Admiralty Arch"
                ],
                "images": [
                ]
           },
           {
                "latitude": 51.49861,
                "longitude": 0.13305,
                "timestamp": 60,
                "descriptions": [
                    "New Scotland Yard",
                    "Metropolitan Police HQ"
                ],
                "images": [
                    "revolvingsign"
                ]
           }
        ]
    }
}
```

Images should be uploaded separately, with a request for each, by POSTing the file and its name to imageUpload.php.

5.4.3.4.1 JSON Schema

Name	Type	Description
authorization	Object, containing one **hash**, and one **salt**	A container to ensure that an upload definitely comes from the server
hash	String	SHA1 of the **salt** combined with a secret passphrase hardcoded into the client and server
Salt	String	Randomly generated to ensure the hash is different every time
Walk	Object, containing one of **title**, **shortDesc**, **longDesc**, and **locations**	A walk is one fully contained route.
Title	String	The title of the route, to be stored in the database
shortDesc	String	The short description of the route
longDesc	String	A long description of the route
locations	Array of **location**	
location	Object, containing a **latitude**, a **longitude**, a **timestamp**, a **descriptions**, and an **images**	One point of interest along the route
latitude	Float	A decimal representation of the latitude of the point of interest
longitude	Float	A decimal representation of the longitude of the point of interest
timestamp	Float	The time since the start of the walk that the point of interest is reached, in minutes
descriptions	Array of zero or more Strings	Descriptions of the point of interest
images	Array of zero or more Strings	The name of images associated with each point of interest, without file extension

5.4.3.4.2 Server Responses

The server should respond with a HTTP header with a status code as defined in RFC 2616. In the event of an error, a JSON formatted error message may be sent to the client, who should display it accordingly. Common HTTP status codes are specified below:

200 OK	The server received the information and all was well.
201 Created	The server received the information and created a new walk.
400 Bad Request	The JSON was not formatted correctly; the walk was not saved.
401 Unauthorized	The authorization failed, and the hashes did not match; the walk was not saved.
404 Not Found	The data was not sent to the correct URL; the walk was not saved.
500 Internal Server Error	The server encountered a problem; the walk was not saved.
503 Service Unavailable	The server is overloaded; try again later; the walk was not saved.

5.4.3.4.3 JSON

Test for config disallowing uploads	400	Uploading has been disabled by the system administrator.
Disallow GETs	405 Method Not Allowed	Only POST is accepted.
Catch empty POST requests	400	No POST variables were sent.
POST needs a key to send its data within key-value pairs. The key "data" has been chosen.	400	Data was not POSTed correctly.
Check if JSON decode failed; PHP does this by returning null on failure, and then has another function to see what the error was	400	Unable to parse JSON.
Verify authorization / authorization section exists	400	No hash present. / No salt present
SHA1 is 40 chars long when expressed in hex	400	Hash is wrong length.
Hashing is case sensitive; as PHP returns lowercase. we need to ensure the user's hash is also lowercase	400	Invalid credentials.
This will be shown when there are no credentials found.	401 is not applicable as we are not using authentication headers	No credentials found.
Check there is actually data	400	No route found.
We are enforcing that no properties are optional this does not check whether the properties contain anything And instead checks merely whether they exist.	400	Route has no title. Route has no subtitle. Route has no description. Route has no locations.
Check that all mandatory properties exist	400	Location $index has no latitude. Location $index has no longitude. Location $index has no timestamp. Timestamp $index is not numeric. Location $index has no descriptions. Location $index has no images. Location $index has no name.
Sanity checks some latitude is drawn from the equator (0) to the poles at 90 and -90.	400	Latitude $index is out of bounds.
longitude is east-west, and is therefore 0 (Greenwich) to # +-180, which are equivalent.	400	Longitude $index is out of bounds.
If a timestamp is not numeric	400	Cannot parse timestamp $index.
If a location-description is not an array	400	Location $index has bad descriptions (not an array).
If a description is not in an array	400	One or more descriptions are not in an array.

If the description array is “!=2”	400	One or more description arrays are malformed.
If a location contains bad images	400	Location $index has bad images (not an array).
If an image tied to a location is not in an array	400	Non-arrays are not in the image array.
If the image array is “!=2”	400	One or more image arrays are malformed.
simplistic way of checking if name has an extension	400	$image [0] does not appear to have a file extension.
If an image is equal to another image	400	File name collision; $image [0] already exists and we won't clobber.
If an image has the wrong encoding	400	$image [0] has bad base64 encoding; check and try again.
Server error	500	File put failed; server error", true.
Default server error	500	File upload failed; no further information available.
Saving to the database failed. This is a server error, not a client issue, so set 500.	500	Unable to save to database: $dbInput".

6. BIBLIOGRAPHY

Help with globe logo
http://24p.com/wordpress/index.php?cat=5&paged=3 accessed 22/01/2014

Help with icon
http://www.iconarchive.com/show/plump-icons-by-zerode/Help-Support-icon.html accessed 22/01/2014

Help with Button null pointer:
http://stackoverflow.com/questions/20957891/android-why-is-this-button-satying-as-null-when-assigning-its-value-based-off-f accessed 24/01/2014

GPS tracker:
http://stackoverflow.com/a/15757944/2942536 accessed 24/01/2014

Photo help
http://developer.android.com/training/camera/photobasics.html accessed 06/01/2014

Timer help:
http://stackoverflow.com/questions/21316366/android-how-can-i-update-a-view-element-from-a-private-inner-class-which-is-an accessed 27/01/2014

Timer problem:
http://stackoverflow.com/questions/21319364/android-how-to-stop-my-timer-correctly-display-the-hhmmss-they-carry-on-counti accessed 27/01/2014

base64 sting bitmap
http://stackoverflow.com/q/14926005/2942536 accessed 29/01/2014
http://mobile.cs.fsu.edu/converting-images-to-json-objects/ accessed 29/01/2014

JSON error messages
http://github.com/cs22120/server/upload.php accessed 29/01/2014

Distances between points
http://forums.asp.net/t/1952508.aspx?Draw+lines+between+markers+on+Google+Map+ accessed 30/01/2014

Map workings and Google APIs
https://developers.google.com/maps/articles/phpsqlajax_v3 accessed 28/01/2014

QA Document SE.QA.03 - General Documentation Standards.

QA Document SE.QA.08 - Operating Procedures and Configuration Management Standards.

DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to document	Changed by
0.1	N/A	2014-12-03	Collated document	bar5
0.2	N/A	2014-12-06	Additional collating	bar5
0.3	N/A	2014-01-27		ays8
0.4	N/A	2014-01-29	Formatting, spelling and grammar checks and adding additional content and bibliography	bar5