Group Project Reserve Plant Species

February 16

2015

Maintenance Report

Group 10

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

Authors:

Mohannad Zeido (moz1@aber.ac.uk)
Christopher Malton (cpm6@aber.ac.uk)

Contents

1 Introduction

In this document we are going to go over all of the required information deemed important enough to be a noteworthy to someone maintaining or making changes to the program.

1.1 Scope

The scope if this document is to provide all the information we have deemed informant to those that are going to be maintaining or making changes to the program.

1.2 Objectives

The objectives of this document are as follows:

1.2.1 Program Description
1.2.2 Program Structure
1.2.3 Algorithms
1.2.4 Main Data Areas
1.2.5 Files
1.2.6 Interfaces
1.2.7 Suggested Improvements
1.2.8 Things to watch for when making changes
1.2.9 Physical limitations
1.2.10 Rebuilding and Testing

2.1 Program Description - Java

An android application designed to allow the user to record a catalogue of plants, including their associated location (as GPS co-ordinates) and rarity, within nature reserves. The app allows the user to take photos of the plant and surrounding location, before allowing further recordings to be made. Once a set of recordings (corresponding to a specific visit to the nature reserve) is complete, the app uploads the information to a web database once an internet connection is available.

2.2 Program Description - Website

The Web App has been created using HTML, CSS and PHP with all the data (recordings, reserves & species) being stored in a MySQL database. The Main home page of the Web App explains the purpose and usage of the website in order to give the user a general overview of its functionality.

The Web App also has a authentication system where an authorized user can log in to the website in order to gain additional functionality if the username and password they have entered is correct. The websites login username and password is stored within variables in the login PHP file.

3.1 Program Structure - Java

This Android application uses a number of modules to gather, manage and send data for sighting records. The list of the modules used is shown below.

U-I modules:

- **Main Activity,** main screen that will open up when the application is run. Shows data recorded so far if available and allows adding, editing, deleting and sending of data.
- **Log On Dialog Fragment,** modal window that allows the entry of user contact details.
- **Reserve Entry Dialog Fragment,** modal window that allows the entry/identification of the reserve.
- **Sighting Entry Dialog Fragment,** modal window that allows the entry of sighting details including GPS coordinates and location and specimen photos.
- **Sighting List View,** list view that allows scrolling and clicking of a list of sightings in the current recording.
- **Sighting Edit Dialog Fragment,** modal window that allows re-entering sighting details to replace a sighting record.
- **Menu in Main Activity,** menu that holds a number of options to edit user details and reserve and to send the whole recording to the server or deleting it.

Data modules:

- **User,** contains user's contact details.
- **Sighting,** contains details for a sighting.
- **Visit,** contains one user and a number of sightings.

HTTP modules:

- **Send To Server,** uses HTTP post to send the recordings to the server.

These modules will be explained in more detail on the following pages.

Main Activity (creation)

This section explains the steps for the creation of the main activity.

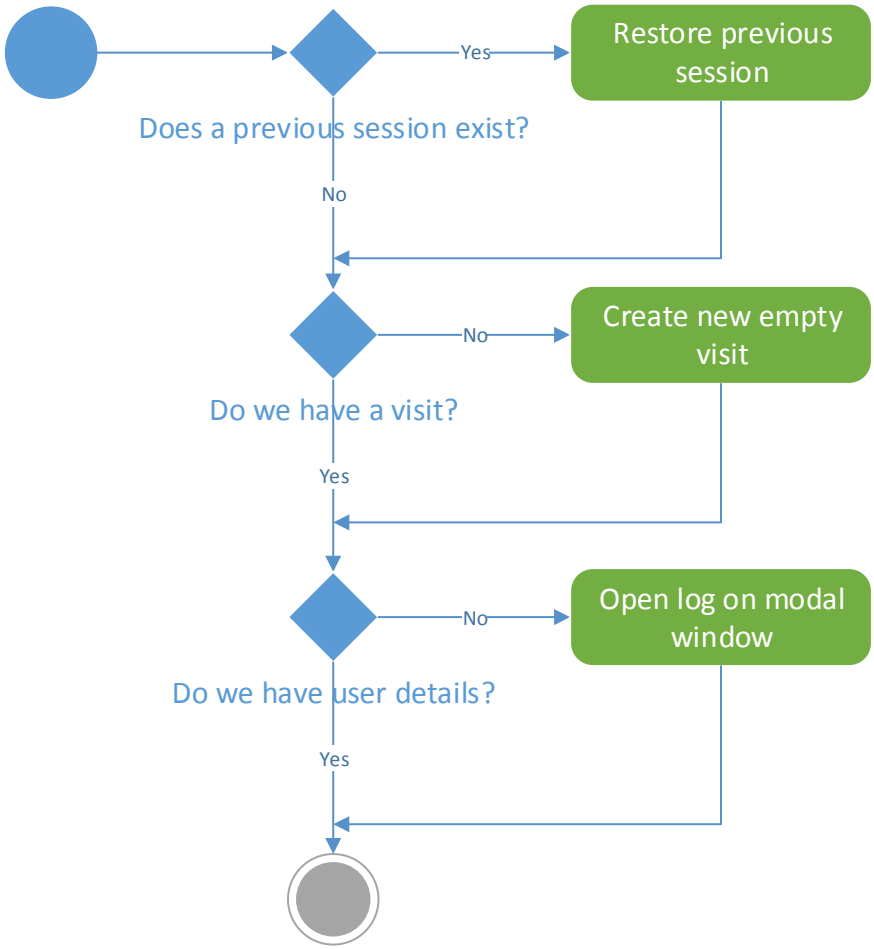

Diagram 1 (UML Activity diagram showing the control flow of the creation of the main activity)

An activity can get destroyed after the user presses the back button or the Android OS has unallocated its memory. Therefore the main activity will always try to restore a previous session. The previous session will be stored by overriding the onSaveInstanceState(Bundle savedInstanceState) method which parcels the current data in the application.

If on creation it did not restore a previous session it will not have a visit and therefore needs to create a new one.

If on creation it did have a visit from a previous session, but no user (contact details), it will create a new one. For this a modal window is opened which will be explained next.

Finally it will create an on item click listener to be used for the list view that contains all the sightings of the current visit.

3.1.1 Log On Dialog Fragment

This section explains how the log on dialog fragment works and how data is passed between it and the main activity.

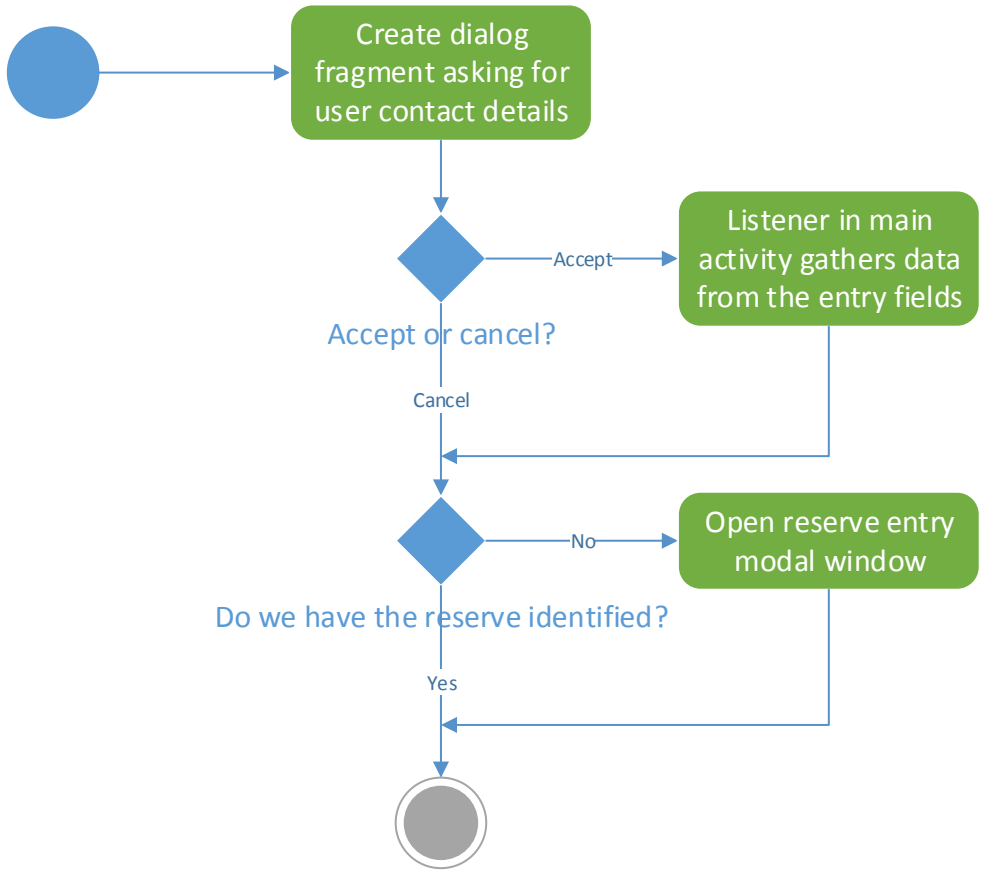

Diagram 2 (UML Activity diagram showing the control flow of the log on dialog fragment)

The dialog fragment is created from within the main activity. This means that the main activity does not get destroyed, it stays behind the dialog (modal window).

The main activity implements a listener that will be used by this dialog to call certain methods.

While building the dialog fragment a view gets created which can be passed through the listener methods as a parameter. This view will make it possible for the main activity to access the elements on the dialog fragment.

If the entry is cancelled due to the user pressing “cancel” or tapping outside of the modal window (and therefore closing it), no new user will be created.

After accepting or cancelling this dialog the main activity checks if a reserve has already been identified. If not it will open a new modal window which will be explained next.

3.1.2 Reserve Entry Dialog Fragment

This section explains how the reserve entry dialog fragment works and how data is passed between it and the main activity.

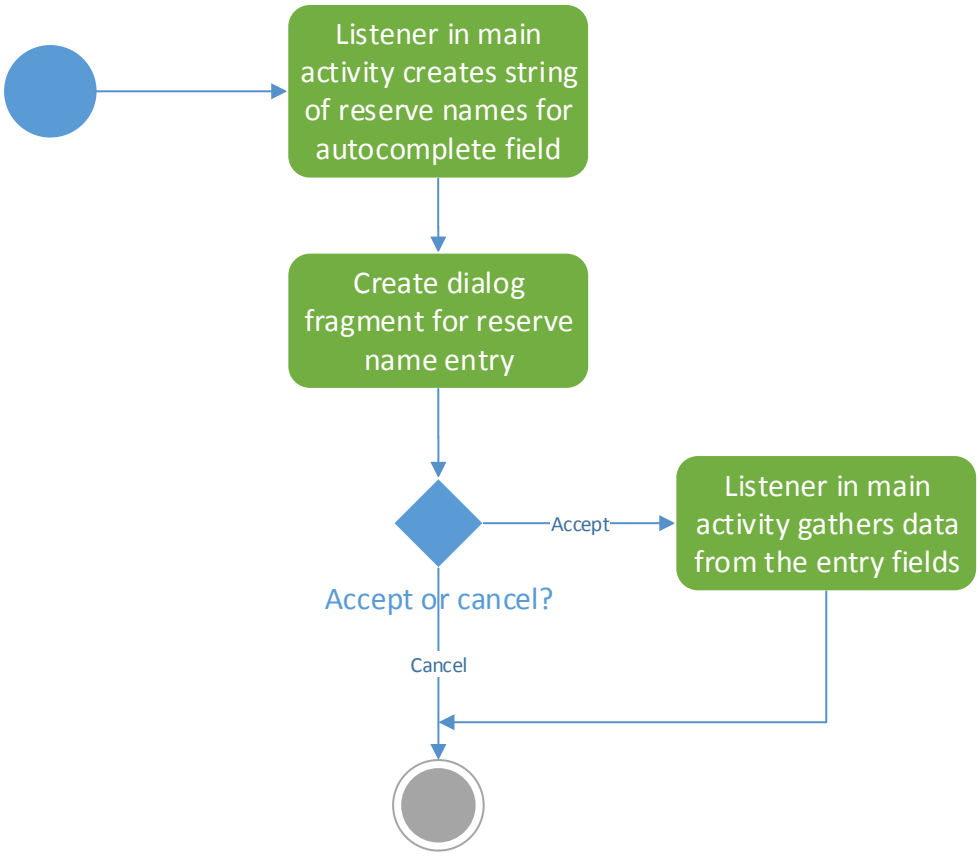

Diagram 3 (UML Activity diagram showing the control flow of the reserve entry dialog fragment)

This dialog fragment works the same as the log on dialog fragment explained before.

It does contain an autocomplete field to enter the reserve name. This field uses an array of strings which gets loaded in the main activity. A method of the listener in the main activity will be used at the creation of this dialog using the view for the connection between the two.

The main activity will get the reserve names from the strings file build into the application and create an array with them. This array will be given to the autocomplete field in the dialog fragment.

After the reserve has been identified, the main activity will update the text field showing the name of the reserve for the current visit.

3.1.3 Sighting Entry Dialog Fragment

This section explains the steps after pressing the "add sighting" button on the main activity and how the sighting entry dialog fragment works and how data is passed between it and the main activity.

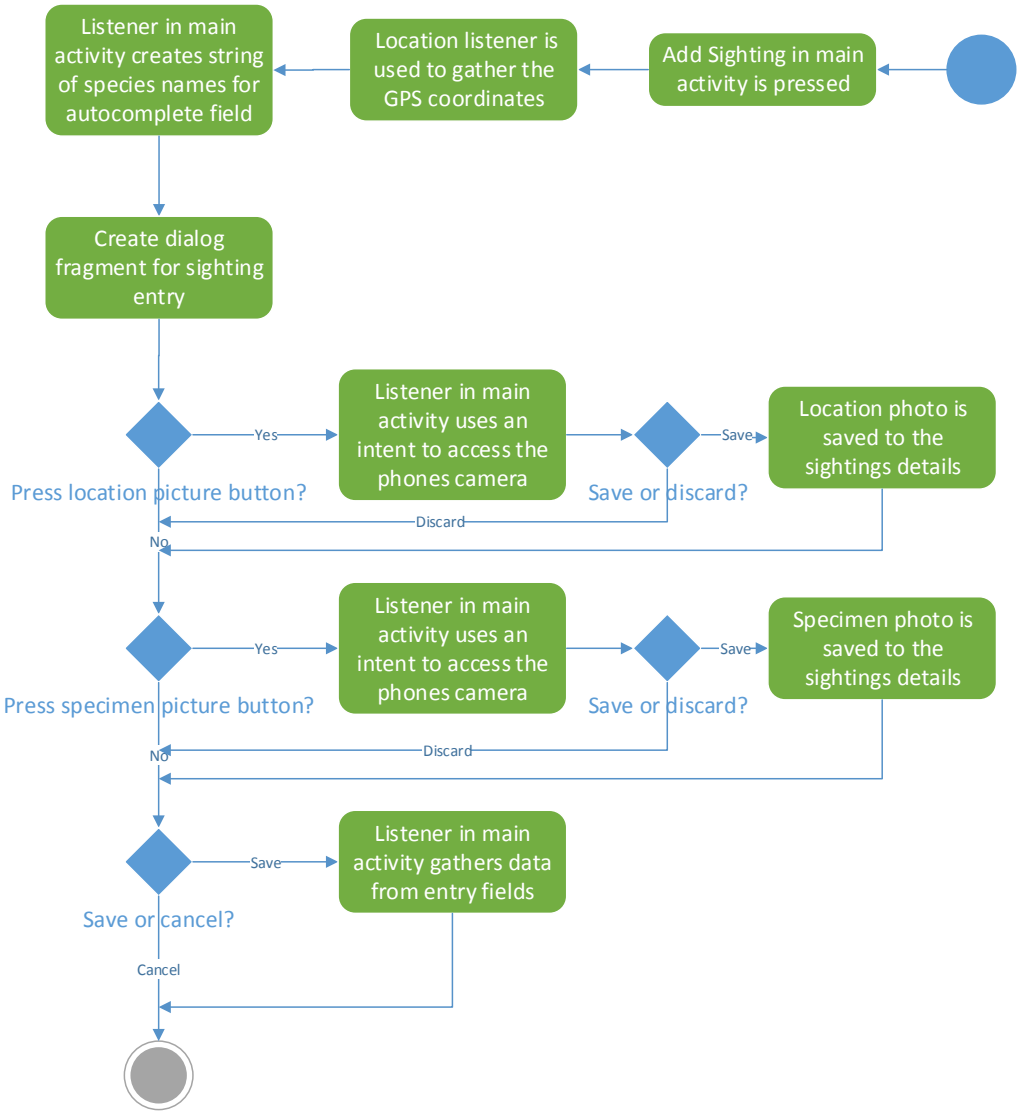

Diagram 4 (UML Activity diagram showing the control flow of the sighting entry dialog fragment)

When the "add sighting" button is pressed the location listener in the main activity will try to retrieve the GPS coordinates from the phone. If this succeeds it will store the longitude and latitude values until sighting entry is finished by the user and a new sighting will be created using these values.

The sighting entry dialog fragment works the same way as the reserve entry dialog fragment explained before. It also uses an array of species gathered from the string file on the application for an autocomplete field.

On the dialog fragment there are two buttons that allow taking photos for the specimen and its general location. When these buttons are pressed the listener in the main activity will use an intent in order to use the phone's camera.

Just like the location, the pictures taken will be saved in the main activity until the sighting is finished.

After saving or cancelling the data entry, the location and photos stored in the main activity will be reset.

3.1.4 Sighting List View (create/update)

This section explains how the list view is managed using an adapter.

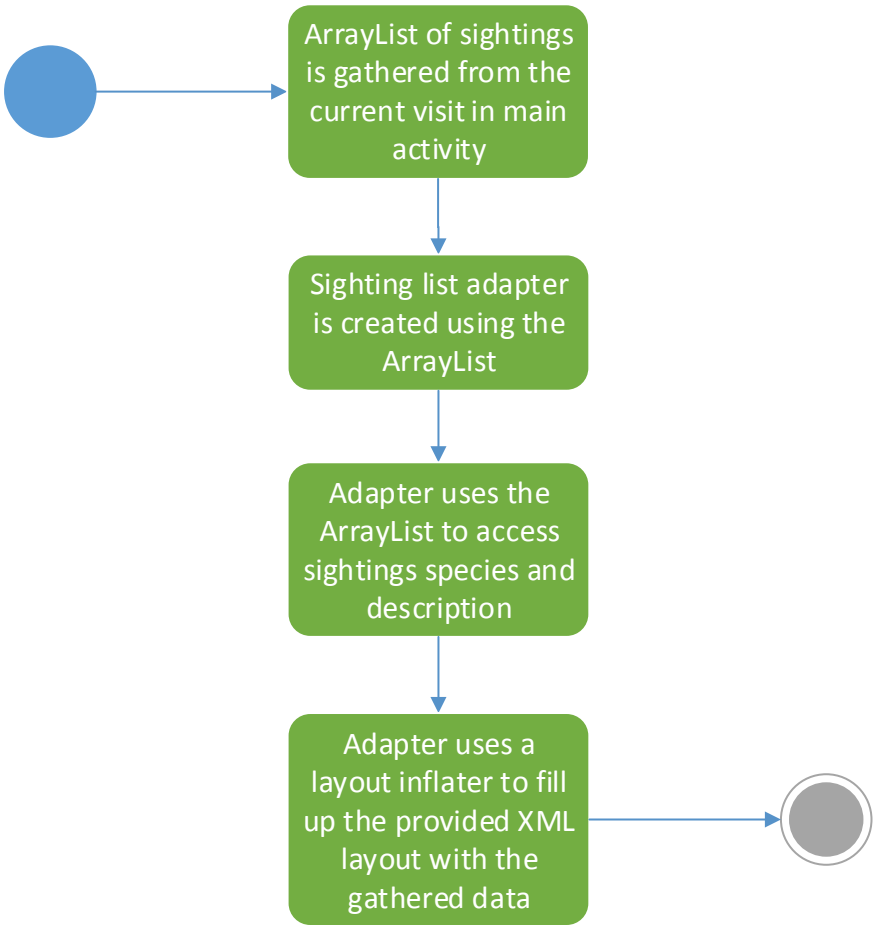

Diagram 5 (UML Activity diagram showing the control flow for the creation of the sighting list view)

The main activity contains a list view which will use an adapter to show the list of sightings in the current recording. The diagram above shows the steps made to create the adapter and how it fills the list with the necessary data.

The list view has to be updated every time the sighting list is altered. At the end of successfully adding, replacing or deleting a sighting the method updateSightingList() will go through these steps to refresh the list.

3.1.5 Sighting Edit Dialog Fragment

This section explains how the available sightings in the list view are accessed to be edited or deleted using the on item click listener.

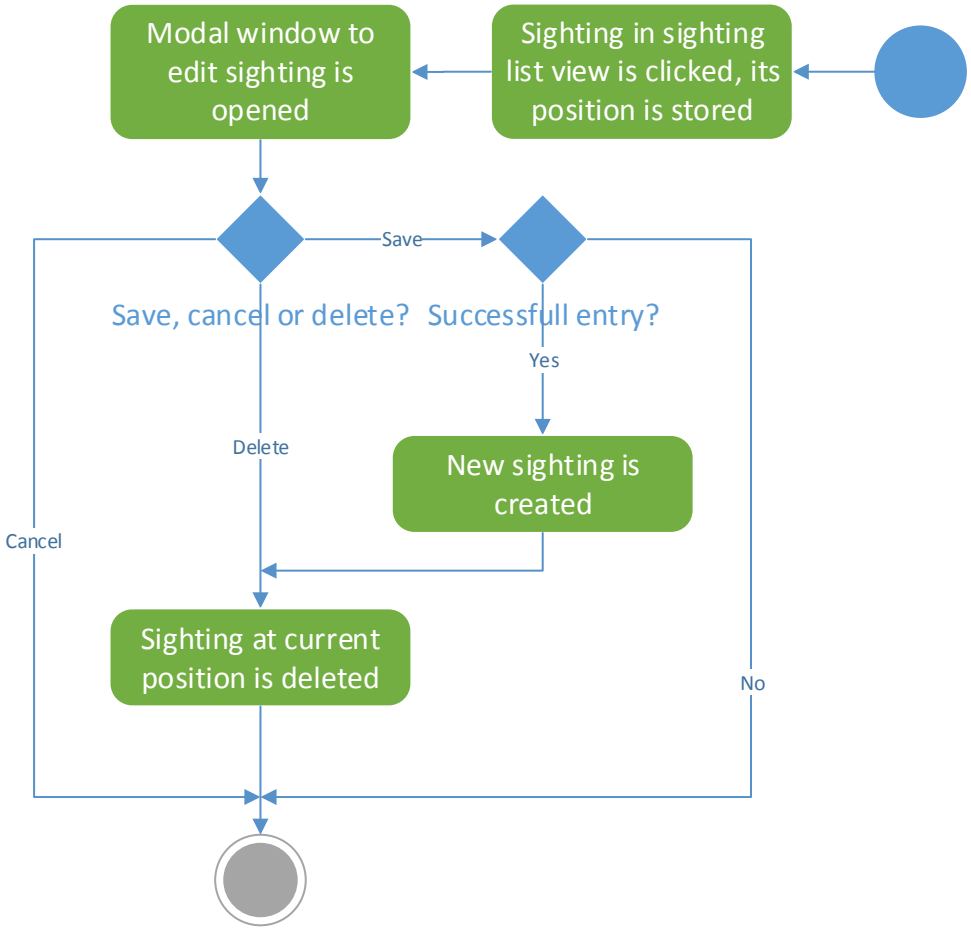

Diagram 6 (UML Activity diagram showing the control flow for editing and deleting current sightings)

An instance of the onClickListener set to the sighting list view when the main activity is created. This listener will get the position of the item clicked and store it in the main activity.

It will then open a new dialog fragment that allows re-entering the data for the current sighting. This dialog will reuse the XML layout of the sighting entry dialog fragment. It will also have a delete button that will delete the sighting marked by the sighting's current position in the main activity.

The save button will use the same method used for the sighting entry, it will also use the Boolean value returned and delete the current sighting (which will be replaced) only if the new sighting is created successfully.

After saving, deleting or cancelling the sighting edit, the current position will be reset.

3.1.6 Menu in Main Activity

This section explains the options available in the menu on the main activity.

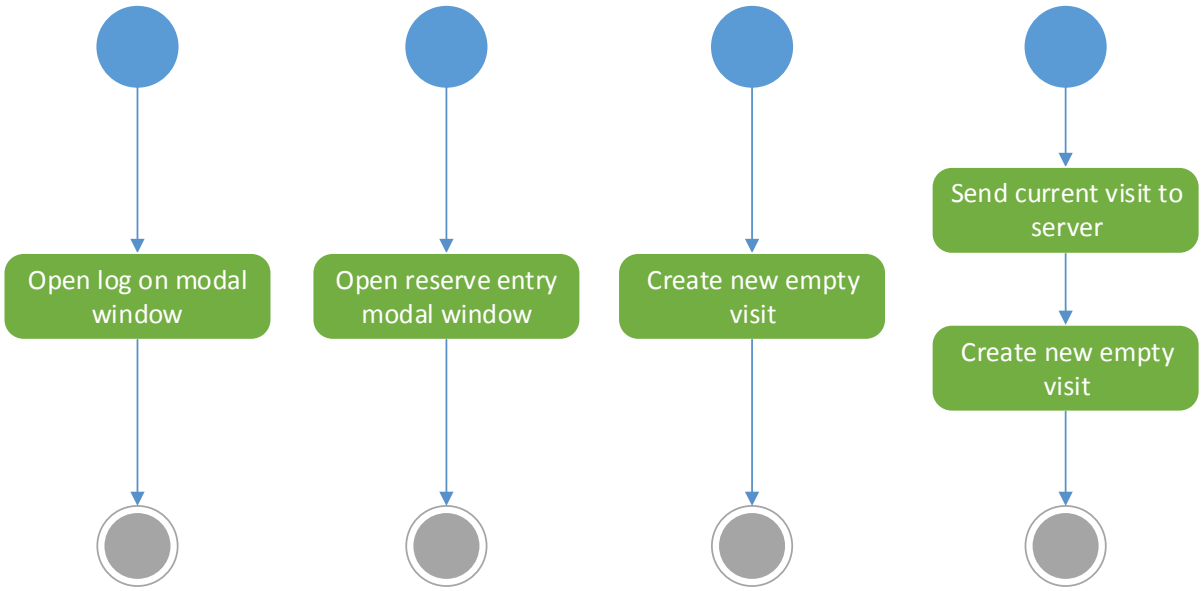

Diagram 7 (UML Activity diagram showing the control flow for the four different options in the main activity menu. Starting from the left, Edit User Contact Details, Edit Current Reserve, Delete whole Recording and Send Recording)

The main activity has a menu, typically accessed by a button on the top right of the menu bar. This menu holds four different options.

The first option will create a log on dialog fragment as explained before.

The second option will create a reserve entry dialog fragment as explained before.

The third option will create a new empty visit and therefore delete the old visit with all its sightings.

The last option will send the visit (as explained before). It will than create a new empty visit and therefore delete the old visit with all its sightings.

3.1.7 Methods used in Main Activity

This section explains the steps after pressing the “send visit” button in the menu list and how the data is sent and what happens to the data on the phone after it is sent.

Once the send visit menu button is pressed the constructor of the send to server class is called, which receives the visit to send, and begins the sending process in an AsyncTask thread. This thread will handle everything related to sending data to the server.

The AsyncTask thread is implemented as a private class in the Send To Server class. The thread is started from the constructor of the Send to Server class which initialises the default constructor and the calls the execute method, which is a default method for the AsyncTask.

The execute method calls the doInBackground method which then starts the process to add the data to the HttpPost. It calls the sendData method which requires a visit object as its parameter. In this method there is a for loop which will repeat the sending process for each sighting that needs to be sent. Each sighting that is going to be sent is stored in an HttpPost which contains the following data is sent: naturalist name, email, phone number, reserve name, date and time of visit, species name, abundance, description, and GPS location.

Once the visit is sent, the data is then deleted from the main activity and the user could start to record a new visit.

Methods used privately in Main Activity

protected void onCreate(Bundle savedInstanceState);
Method that runs when the main activity gets created. It will do the steps explained in Main Activity (creation) using savedInstanceState to recover any previous records.

public void onItemClick(AdapterView<?> parent, View view, int position, long id);
Method used in the onItemClickListener used for the sighting list view. It will use int position to determine what item was selected in the list.

public boolean onCreateOptionsMenu(Menu menu);
Method that runs when the main activity gets created. It will populate the menu in the main activity.

public boolean onOptionsItemSelected(MenuItem item);
Method that runs when an item (MenuItem) inside the main activity's menu is selected. It will do the steps explained in Menu in Main Activity.

public void onSaveInstanceState(Bundle savedInstanceState);
Method used to parcel the current recording into the savedInstanceState bundle when the main activity gets destroyed. The recording can be recovered when rebuilding the main activity.

public void addUser();
Method that creates the log on dialog fragment and shows it on screen.

public void selectReserve();

Method that creates the reserve entry dialog fragment and shows it on screen. It will also call addDate() to add a date to the current visit.

public void addDate();
Method that uses an instance of a calendar to get the current date and time. It will then reformat it to *"yy-MM-dd HH:mm:ss"* and store it in the current visit's date string.

public void updateLocation();
Method used to reset the text view in the main activity showing the current reserve name.

public void recordSighting(View view);
Method that creates the sighting entry dialog fragment and shows it on screen.

public void updateSightingList();
Method that gets the current list of sightings in the recording and creates an adapter that populates the sighting list with these sightings.

public void editSighting();
Method that creates the sighting edit dialog fragment and shows it on screen.

Methods accessed by the Log On Dialog Fragment

public void onLogOnDialogPositiveClick(DialogFragment dialog);
Method that will access the log on dialog fragment elements using the parameter dialog to gather the information entered by the user and create a new user instance.

public void onLogOnDialogNegativeClick(DialogFragment dialog);
Method that cancels the user instance creation.

Methods accessed by the Reserve Entry Dialog Fragment

public void onCreateReserveSearch(View view);
Method that gathers the list of reserve names and uses an adapter to populate the autocomplete text field on the dialog fragment, accessing it using the parameter view.

public void onReserveEntryDialogPositiveClick(DialogFragment dialog);
Method that will access the reserve entry dialog fragment elements using the parameter dialog to gather the information entered by the user and store the reserve identified.

public void onReserveEntryDialogNegativeClick(DialogFragment dialog);
Method that cancels the identification of the reserve.

Methods accessed by the Sighting Entry/Edit Dialog Fragments

public void onCreateSightingSearch(View view);

Method that gathers the list of species names and uses an adapter to populate the autocomplete text field on the dialog fragment, accessing it using the parameter view.

public void onCreateGetLocation(DialogFragment dialog);
Method that uses a location manager to get the latitude and longitude for the last known location of the phone.

public boolean onSightingEntryPositiveClick(DialogFragment dialog);
Method that will access the sighting entry/edit dialog fragment elements using the parameter dialog to gather the information entered by the user and create a new sighting instance. Returns value true if the creation of the sighting was successful.

public void onSightingEntryNeutralClick(DialogFragment dialog);
Method that cancels the entry/editing of a sighting.

public void onSightingEditNegativeClick(DialogFragment dialog);
Method that deletes the current sighting selected on the sighting edit dialog fragment.

public void onCameraSpeciesClick(DialogFragment dialog);

public void onCameraLocationClick(DialogFragment dialog);

private void dispatchTakePictureIntent();

protected void onActivityResult(int requestCode, int resultCode, Intent data);

public void onLocationChanged(Location location);
Method that gets the longitude and latitude values from the parameter location and stores them in two separate variables.

public void onStatusChanged(String provider, int status, Bundle extras);
Method implemented for the location listener.

public void onProviderEnabled(String provider);
Method implemented for the location listener.

public void onProviderDisabled(String provider);
Method implemented for the location listener.

3.1.8 Methods used in Log On Dialog Fragment

public Dialog onCreateDialog(Bundle savedInstanceState);
Method that runs when the log on dialog fragment is created. It will set its title and create the positive and negative buttons.

Method used by positive button

public void onClick(DialogInterface dialog, int id);
Method that passes the current dialog on to the main activity in the parameter of the onLogOnDialogPositiveClick(DialogFragment dialog) method in the LogOnDialogListener.

Method used by negative button

public void onClick(DialogInterface dialog, int id);
Method that will call the onLogOnDialogNegativeClick(DialogFragment dialog) method in the LogOnDialogListener.

public void onAttach(Activity activity);
Method that runs when the log on dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the LogOnDialogListener and throw an error if the parent activity has not implemented it.

3.1.9 Methods used in Reserve Entry Dialog Fragment

public Dialog onCreateDialog(Bundle savedInstanceState);
Method that runs when the reserve entry dialog fragment is created. It will set its title and create the positive and negative buttons.

Method used by positive button

public void onClick(DialogInterface dialog, int id);
Method that passes the current dialog on to the main activity in the parameter of the onReserveEntryDialogPositiveClick (DialogFragment dialog) method in the ReserveEntryDialogListener.

Method used by negative button

public void onClick(DialogInterface dialog, int id);
Method that will call the onReserveEntryDialogNegativeClick (DialogFragment dialog) method in the ReserveEntryDialogListener.

public void onAttach(Activity activity);

Method that runs when the reserve entry dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the ReserveEntryDialogListener and throw an error if the parent activity has not implemented it.

3.1.10 Methods used in Sighting Entry Dialog Fragment

public Dialog onCreateDialog(Bundle savedInstanceState);
Method that runs when the sighting entry dialog fragment is created. It will set its title and create the positive and negative buttons. It will also call the onCreateGetLocation(DialogFragment dialog) and the onCreateSightingSearch(View view) methods passing the current dialog and the view as parameters respectively.

Method used by positive button

public void onClick(DialogInterface dialog, int id);
Method that passes the current dialog on to the main activity in the parameter of the onSightingEntryPositiveClick (DialogFragment dialog) method in the SightingEntryListener.

Method used by neutral button

public void onClick(DialogInterface dialog, int id);
Method that will call the onSightingEntryNeutralClick (DialogFragment dialog) method in the SightingEntryListener.

public void onAttach(Activity activity);
Method that runs when the sighting entry dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the SightingEntryListener and throw an error if the parent activity has not implemented it.

3.1.11 Methods used in Sighting Edit Dialog Fragment

public Dialog onCreateDialog(Bundle savedInstanceState);
Method that runs when the sighting edit dialog fragment is created. It will set its title and create the positive and negative buttons. It will also call the onCreateSightingSearch(View view) method passing the current view as parameter.

Method used by positive button

public void onClick(DialogInterface dialog, int id);
Method that passes the current dialog on to the main activity in the parameter of the onSightingEntryPositiveClick (DialogFragment dialog) method in the SightingEntryListener (reusing this listener).

Method used by neutral button

public void onClick(DialogInterface dialog, int id);
Method that will call the onSightingEntryNeutralClick (DialogFragment dialog) method in the SightingEntryListener (reusing this listener).

Method used by negative button

public void onClick(DialogInterface dialog, int id);
Method that will call the onSightingEditNegativeClick (DialogFragment dialog) method in the SightingEditListener.

public void onAttach(Activity activity);
Method that runs when the sighting entry dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the SightingEntryListener and the SightingEditListener and throw an error if the parent activity has not implemented them.

3.1.12 Methods used in Sighting List Adapter

public SightingListAdapter(Context context, ArrayList<Sighting> values)
Constructor for the adapter that takes an array list of sightings that is used to populate the list view.

public View getView(int position, View convertView, ViewGroup parent)
Method that uses an XML layout for the list view items and uses the provided sighting list to fill in the data species name and description.

3.1.13 Methods used in Send to Server class

public SendToServer(Visit visit);
Method which is the constructor for this class. The parameter it requires is a visit. In this method the global variable visit is set to the visit which has been passed in as the parameter. This method also instantiates the private AsyncDownloader class by calling the default constructor. The final thing this method does is initialises the sending thread.

Methods used in the AsyncDownloader private class

protected Integer doInBackground(Object... params);
Method calls the sendData method. This method needs to be implemented as part of the inheritance from the AsyncTask class. The parameter is not used as this class is called as part of the execute method behind the scenes. It returns 1 meaning the send is completed.

public boolean sendData(Visit visit);
Method requires a visit as its parameter. This method initialises an HttpClient as a DefaultHttpClient. It then iterates through the array of sightings in the visit and then for each sighting calls the sendUserData. It returns a boolean but that is not currently used in our code, but it would be used to indicate weather the method has passed or failed.

private boolean sendUserData(Visit, visit, HttpClient client, int sighting);
Method which adds the visit information to the HttpPost. The parameters it requires are a Visit, an HttpClient, and a sighting int. Visit is the information about the visit which is going to be sent. The client is the HttpClient which we use as the default client. The sighting is the index, in the array of sightings, indicating the sighting which is going to be sent. This method sets the target URL for which the data will be sent to. The information for the sighting which is going to be sent is packaged as an arrayList of key value pairs. The arrayList is then added to the HttpPost as an entity. Finally the sendFinalData is called to send the data. It returns the result of that call which would be used to check if the data was sent successfully.

private boolean sendFinalData(HttpClient client, HttpPost httppost);
Method executes the final send command to send the data in the HttpPost to the database. The parameters it requires are HttpClient, and HttpPost. These are both passed in from the previous method. This method contains some error checking but we did not have enough time to implement it in the rest of the code. The client execute call returns a response which contains the state of the data sent. A successful send would return the decimal number 200.

private HttpPost createNewPost();
This method was meant to reduce the repetition in the code when more than one HttpPost will be used, so instead of declaring them many times, this method will make it easier and cleaner. In the end we did not use it.

3.2 Program Structure - Website

It is possible to view a list of all the recordings made (sent from users of the android app) by navigating to the "recordings" page of the website. On this page it is also possible to view only recordings made at a specific reserve by using the drop down menu provided, this search functionality is achieved by using an sql statement which retrieves only recordings which match the specified criteria from the database. The list of recordings can also be sorted in order based on the species name or the date the recording was made which is achieved by using the "ORDER BY" keyword in sql. In addition to all of the above the user can also click on the small thumbnail image of either the scene or specimen photo in order to view the full-sized image, this functionality was achieved by using a resource called Lightbox by Lokesh Dhakar (http://lokeshdhakar.com/projects/lightbox2/).

A complete list of all the reserves stored in the database can be viewed by navigating to the "reserves" page of the website, on this page it is also possible to edit, delete or add reserves however these options are only available to authenticated users. The reserves are sorted in alphabetical order for convenience.

To edit or delete a specific reserve there are buttons at the end of each reserve row. Clicking the delete button executes an sql statement which removes the corresponding reserve from the reserves database table. Clicking on the edit button takes the user to an edit reserve page ("edit.php") which displays the current information about the reserve and gives the user the option to change it. The edit page contains field validation code which validates any user input to make sure it is not malicious and is correctly formatted, regular expressions and built-in php functions like "strlen" are used in order to achieve this.

If the user does enter invalid input, a message is shown to the user pointing out the errors and the data is not stored in the database. It is possible to add a reserve by clicking on the "Add Reserve" button again found on the "reserves" page of the website, this works in a similar fashion to the edit functionality however in this case the form fields will not be pre-populated with existing data. The user of the web app can enter information about a reserve they wish to add and this input will be validated before the new reserve is added to the 'reserves' database table.

The recordings data sent from the android app via HTTP Post is received by a page called "fetch_data". This page validates then stores the received recordings data into the 'recordings' database table but only if the validation passes. Every time a request is received information about the HTTP Post request is stored within a log file.

The design of the website is relatively simple which allows ease of use of the application. The navigation of the website is somewhat natural, meaningful names are given to pages based on their function and the required action of the user.

There is a login page for the administrator (user) to use to gain access to the website. Once past the login, the user (administrator) can use the application to create, edit and delete reserves. The recordings page displays all recordings in the database, this page makes use of pagination to organise the data per page making it easier for the user to click through the site whilst avoiding needless scrolling. On the recordings page the user can sort plants by both date and species. A reserve can be selected to filter down a search for recordings of plants specific to that reserve 0selected.

Component diagram can be found in the design specification on page 5

Web sequence diagram can be found in the design specification on page 13

Method/algorithms can be found in the design specification on pages 15 and 16

4.1 Algorithms - Java

Get the sighting's location

The location is needed for every sighting, at the creation of the sighting entry dialog fragment the following code is used:

```
public void onCreateGetLocation(DialogFragment dialog) {
    LocationManager locationManager = (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);

    Criteria criteria = new Criteria();
    String provider = locationManager.getBestProvider(criteria, false);
    Location location = locationManager.getLastKnownLocation(provider);

    if (location != null) {
        onLocationChanged(location);
        Toast.makeText(this, "LAT: " + locLat + " LNG: " + locLng,
            Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Cant find location using " + provider,
            Toast.LENGTH_SHORT).show();
    }
}
```

A location manager is invoked from the Android system services. The location manager is asked for the best provider available, this can be GPS or network. The provider is saved in a string and used by the location manager to get the last known location.

```
public void onLocationChanged(Location location) {
    locLat = location.getLatitude();
    locLng = location.getLongitude();
}
```

If getting the location was successful its longitude and latitude values will be saved in the main activity for later use.

If the location cannot be accessed a toast is shown as feedback.

Opening a modal window

A number of modal windows is used in this application. They are created in the main activity. A listener is used to communicate between the window and its parent activity. The code for the creation of the sighting entry dialog fragment is shown below:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
    LayoutInflater inflater = getActivity().getLayoutInflater();
    View view = inflater.inflate(R.layout.fragment_sighting_entry, null);
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

    builder
        .setView(view)
        .setTitle(R.string.sighting_entry_dialog)
        .setPositiveButton(R.string.save,
            new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                fragmentListener
            .onSightingEntryPositiveClick(SightingEntryFragment.this);
            }
        })
        .setNeutralButton(R.string.cancel,
            new DialogInterface.OnClickListener() {
            @Override
```

```
            public void onClick(DialogInterface dialog, int id) {
                fragmentListener
            .onSightingEntryNeutralClick(SightingEntryFragment.this);
            }
        });

    fragmentListener.onCreateGetLocation(SightingEntryFragment.this);
    fragmentListener.onCreateSightingSearch(view);

    ImageButton specimenPictureButton = (ImageButton) view
      .findViewById(R.id.captureSpecimenImage);
    specimenPictureButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            fragmentListener
            .onCameraSpeciesClick(SightingEntryFragment.this);
        }
    });




    ImageButton locationPictureButton = (ImageButton) view
      .findViewById(R.id.captureLocationImage);
    locationPictureButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            fragmentListener
            .onCameraLocationClick(SightingEntryFragment.this);
```

```
        }
    });

    return builder.create();
}
```

The creation starts by making a view using a layout inflator with a provided XML layout. Than an alert dialog builder is used to build the dialog fragment and set some parameters, including its design (the view), the title and two buttons.

The buttons use a listener implemented by the parent activity to run methods when these are pressed. These methods are given the current dialog fragment, using this the parent activity can use find view by id to access the dialog's elements.

The method onCreateSightingSearch(View view) requires to be given a view, as the autocomplete text field cannot be found using the dialog fragment.

On click listeners are used for the buttons to take photos. These also require a view to find the actual button on the modal window.

Adapter for the autocomplete text fields

The reserve and sighting modal windows contain text fields that autocomplete the reserve and species names respectively. The autocomplete text fields need an adapter that provides the layout and the data. The following code is used for the reserve name autocomplete text field:

```
public void onCreateReserveSearch(View view) {
    Resources r = getResources();
    String [] list = r.getStringArray(R.array.reserves_names);

    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
      android.R.layout.simple_list_item_1, list);

    AutoCompleteTextView textView = (AutoCompleteTextView)
      view.findViewById(R.id.reserveName);
```

```
    textView.setThreshold(1);

    textView.setAdapter(adapter);

}
```

An array of strings is created with the resources stored in the array file of the application. A standard adapter is created for strings using a standard layout and the array of strings as data.

The autocomplete text view is located by its id and the created adapter is set as its adapter.

4.2 Algorithms – Web

See Web Algorithms section in the Design Document.

5.1 Main Data areas - Java

Visit class:

String: 'reserve' - Contains the reserve name that was visited.
String: 'date' - Contains the date the visit took place.
ArrayList<Sighting>: 'sightings' - Contains a list of all different sightings the user input for each visit. A new array list is created to be sent to the server every time.

Sighting class:
String: 'species' - Contains the species type of the sighting.
String: 'abundance' - Information about how rare the plant species appeared to be in the sighting
String: 'description' - A brief description given by the user regarding the plant.
Double: 'locLat' - The latitude of the plants position.
Double: 'locLng' - The longitude of the plants position.
BitMap: 'specimenPicture' - A bitmap image taken by the user of the specimen.
BitMap: 'locationPicture' - A bitmap image taken by the user of the area surrounding the specimen.
Integer: 'specimenPictureState' - Set to '1' if a picture was taken and does exist. Effectively a Boolean style variable.
Integer: 'locationPictureState' - Set to '1' if a picture was taken and does exist. Effectively a Boolean style variable.

User class:
String: 'name' - The users name.
String: 'phone' - The users contact phone number.
String: 'email' - The users contact email address.

**All classes implement Parcelable, which allows data to be transferred between Android Activities.*

5.2 Main Data areas - Web

See Significant Data Structures for the Web App in the Design Document.

6.1 Files - Java

The Java program did not make use of any special files.

6.2 Files - Web

When the Web App receives a HTTP Post Request a log file containing details of the request is created. If the log file already exists however, future requests will be appended to the bottom of the file. The information in these logs is useful as it displays any errors that may have occurred during the receipt of the data.

7.1 Interfaces - Java

See Design Specification for Java Interfaces used.

7.2 Interfaces - Web

See Design Specification for Web Interfaces used.

8.1 Suggested Improvements - Java

More compatibility

The android app only runs on 4.0 to 4.4.2, this restricts the software to newer phones only and not older phones running android 2,3 and newer versions like 5. The lower versions are still used by a lot of people and they are lost potential customers.

"Success! Record sent"

There is no after prompt stating the record is sent. A simple toast message saying “Success! Record sent" tells the user that the record has been sent to the database and then they can safely move on to another recording session.

GPS always shown

A not so needed feature but a feature that shows the exact GPS co-ordinates of the location on the top of the screen if the person has location on. If they want to write it down on paper for later use.

Light Function

A feature useful if the user is working at night and cannot see what they are looking at on the screen. The light provides the ability to see the site and species at night.

8.2 Suggested Improvements - Web

Although the overall functionality of the website is very good, and compliant with W3C standards for browser requirements etc. there are always room for improvement. There are shortcomings that may have arisen due to time constraints, others that did not become apparent until the website was tested or viewed by various people.

To a first time user, the home page may seem like it is somewhat clustered has too much to read. The welcome text could be a bit larger to stand out more and to not make it seem as if it is part of the *Resources Used* heading which uses the same font size. The resources used heading could be moved to another area of the page, perhaps on an *About* page, or lower and in a smaller font than the rest of the text on the same page.

By default, the records are sorted alphabetically (a-z) by Species on the *Recordings* page, yet the species column is the fourth column on the table, to a visitor it might appear that the records are randomly scattered and give the website an unprofessional look. This issue can easily be resolved by making the species column the first column on the table.

The date and time could have been displayed somewhat nicer with more space between the two lines. Most people in the UK are used to the DD-MM-YYYY format, so the current format could be changed to cater more for the target audience.

9.1 Things to watch for when making changes - Java

9.1.1 Main Activity

There are a few private attributes in the main activity that could be classified as global variables, as there is only one activity. These variables always need to be reset after being used. A list of these attributes is shown below:

Visit visit, stores the current recording, should be restored ones the recording is send or deleted.

Boolean specimenPic, stores true if the current sighting has a picture of the specimen. Should be reset once the sighting is created and saved.

Boolean locationPic, stores true if the current sighting has a picture of the location. Should be reset once the sighting is created and saved.

Double locLat, latitude for the current sighting. Its value will be used at the creation of the sighting and should be reset afterwards.

Double locLng, longitude for the current sighting. Its value will be used at the creation of the sighting and should be reset afterwards.

9.1.2 Log On Dialog Fragment

Any activity invoking this dialog fragment should implement:
LogOnDialogFragment.LogOnDialogListener

9.1.3 Reserve Entry Dialog Fragment

Any activity invoking this dialog fragment should implement:
ReserveEntryFragment.ReserveEntryDialogListener

9.1.4 Sighting Entry Dialog Fragment

Any activity invoking this dialog fragment should implement:
SightingEntryListener

9.1.5 Sighting Edit Dialog Fragment

Any activity invoking this dialog fragment should implement:
SightingEntryListener
SightingEditListener

9.2 Things to watch for when making changes – Web

Not applicable to the web and database side of the project.

10.1 Physical Limitations – Java

The program is limited to 4.0 of android as its lowest point of entry. About 15 - 25% of users still use lower versions of android which is a missed demo graph for use with the app.

The program can crash at any time and can interrupt sending if does.

The location can malfunction on the phone and corrupt the GPS data

The user can potentially fiddle with the data structures if they open the APK and change the code with the correct tools.

The contact details allow any character to be put in, such as "[:" as a name and 0000000000 as a number

The apps speed is dependant on how many apps are open in the background and the phones speed

The app can freeze if overloaded by something or freeze because of a memory allocation bug

10.2 Physical Limitations – Web

The website does not currently allow the user to click on a certain species in a recording and view all other instances of that recording; this would be a useful function in the website and could be added in future.

The maximum file-size for an image is currently 5mb. Some high definition pictures today are much larger than this and it could lead to some pictures not being displayed by the website due to this limitation.

When viewed on a mobile device, most of the content is not immediately readable. Users may have to zoom in to view text or turn their device onto landscape mode to make it more readable. This is not a major flaw, as the website was intended to be viewed on computer browsers, but is definitely something that can be looked at in the *future*.

On the species record, it is not possible to accept any symbols, however plants names may contain hyphens or other symbols. Therefore any future versions of this should add support for this option.

It is not possible to change whether you want to see the date when the record was created or based on its last modification. This should be addressed in future versions of the website.

The website is fully compliant with any browsers after (and including) Internet explorer 8, but may not be entirely compatible with older browsers. Although this is compliant with the W3C's standards, it could limit users who may still be using browsers older than that.

11.1 Rebuilding and Testing – Java

11.1.1 Developing tools

This Android application was produced using Eclipse with the ADT plugin for easy Android development. The application is built on the Android SDK 4.4.2 (API 19). The appropriate SDK packages should be installed to further develop this application.

11.1.2 Rebuilding

The ADT plugin allows to export signed and unsigned .APK files, this will be required to rebuild and install the application on an Android phone. Most phones will only allow signed applications to be installed. This requires the developer to provide a key for the .APK file. A new key can be generated following the steps the ADT provides. Once the .APK file is created it can be put on the phone using a USB cable or via email. Now to install the application just open the file, accessing it with a file manager.

11.1.3 Testing

There is a testing specifications document provided with this project. It contains a series of tests for the Android application that can be followed. These tests are designed to make sure that the application works as intended, providing the functionality as detailed in the requirements specification. If new functionality is added to the application new tests should be written for it. It also has to be made sure that any old functionality that has been further developed is tested and the tests currently in place are updated accordingly.

11.2 Rebuilding and Testing – Web

When rebuilding the website you will be required to host the website on a server, this can be done locally (lamp, mamp, camp etc…). The pages of the website need to be put into a folder on the server. Optionally you can store the database under a domain name of your choice.

The database will need to be re-set up on a server. All the files that you need are in a folder on the distributed compact disc.

All testing that has been and can be done in regards to the website can be found in the testing specification, test results are displayed in a column (pass/fail), expected results are also detailed in a separate column. If you follow the testing table any problems with the websites functionality or usability will be alerted and detailed.

To add a test, just add a new row to the table and fill out the different columns using similar language and presentation formatting as to the other rows that are already present in the table.

Version	Date	Changes Made	Changed by
1.0	16.02.2015	Original document.	cpm6, moz1