

CS221 Group Project

Design Specification

Project Team	Aldridge, William	Atkins, Max
	Barnes, Alexander	Dart, Nicholas
	Harizanov, Maksim	O'Hare, James
	Pocock, Michael	Tuff, Sebastian
	Wilcock, Daniel	Wilmot, Andrew
Version	1.0	
Status	Release	
Date Published	February 10, 2015	
Reference	SE_O2_DS_00	
Department	Computer Science	
Address	Aberystwyth University	
	Penglais Campas	
	Ceredigion	
	SY23 3DB	

Contents

1	Decomposition Description	4
1.1	Subsystems	4
1.1.1	Android Application	4
1.1.2	Server	4
1.1.3	Database	4
1.1.4	Web API	4
1.1.5	Website	5
1.2	Significant Android Components	5
1.2.1	Significant UI classes	5
1.3	Significant other classes	6
1.4	Significant Website Components	6
1.5	Significant Server Components	7
2	Android Application Design	8
2.1	Decomposition	8
2.2	Interfaces	9
2.2.1	Login Activity Interface	9
2.2.2	Site Choice Activity Interface	11
2.2.3	Species Selector Activity Interface	12
2.2.4	Visit Management Activity Interface	13
2.2.5	Visit Submit Activity Interface	14
2.2.6	User Data Class Interface	15
2.2.7	Visit Data Class Interface	16
2.2.8	Plant DB Interface	17
2.2.9	Specimen Class Interface	17
3	Web Design	18
3.1	Interface	18
3.2	Detailed design	20
3.2.1	config.php	20
3.2.2	plant_sorting.php	20
3.2.3	simple_search.php	21
3.2.4	advanced_search.php	21
3.2.5	edit_plant_record.php	21
3.2.6	delete_plant_record.php	21
3.2.7	upload_image.php	21
3.2.8	regex validation	21
3.2.9	Index.php	21
3.2.10	Header.php	22
3.2.11	footer.php	22
3.2.12	nav.php	22
3.2.13	Edit/add record	22
3.2.14	delete record	22
3.2.15	delete confirmation pop up	23

3.2.16	config.php	23
3.2.17	authenticate.php	23
3.2.18	delete_curl.php	23
3.2.19	edit_specimens.php	23
3.2.20	footer.php	23
3.2.21	get_reseve.php	23
3.2.22	header.php	24
3.2.23	img_curl.php	24
3.2.24	logout.php	24
3.2.25	map.php	24
3.2.26	specimens_curl.php	24
3.2.27	resdelete_curl.php	24
3.2.28	reserves_curl.php	24
3.2.29	specimens_search.php	24
3.2.30	filter.php	24
4	Server Design	26
4.1	Interface	26
4.1.1	AuthenticateAdmin	26
4.1.2	AddRecord	26
4.1.3	AddReserve	27
4.1.4	RemoveSpecimen	27
4.1.5	GetSpecimen	27
4.1.6	UpdateSpecimen	28
4.1.7	AddResource	29
4.1.8	GetResource	29
4.1.9	GetReserve	29
4.1.10	GetReserves	30
4.1.11	UpdateReserve	30
4.1.12	RemoveReserve	31
4.2	Detailed design	31
4.2.1	Diagrams	31
4.2.2	Significant data structures	33
5	Document History	37

1 Decomposition Description

1.1 Subsystems

The Botanist Tools application is composed of 3 subsystems:

- Android Application
- Website
- Server

1.1.1 Android Application

The android application provides the interface that users will use to record plant data when out on a visit. It implements requirements (FR1), (FR2), (FR3), (FR4), (FR5), (FR6). It must also conform with the requirements (EIR1), (PR1), (PR2), (DC1) and (DC2) [1].

The application will have a form-based activity which can be used to add and edit new and currently saved recordings. Currently saved recordings will be stored locally in a collection, which will be awaiting dispatch to the server. The user will be able to view this list and select recordings to perform actions on (Eg. edit and delete). The application will not show recordings that are saved on the server.

The Android application will communicate with the server to perform functions such as sending recordings and performing user authentication. The application does NOT communicate directly with the website and the database. It uses a web API which is core to the server.

1.1.2 Server

The server will consist of two parts:

- A database
- A web API

1.1.3 Database

The database will be the central datastore for the entire system. It will communicate exclusively with the web API and serve as its back-end.

1.1.4 Web API

The web API is central to the system; it provides a uniform way of accessing the database for all subsystems to use. It maintains the integrity of the datastore by acting as the “middle-man” so that the other subsystems do not damage the contents. It exposes a public interface to allow a set of actions to be performed by the users of the API; actions include user authentication and recordings management. The web API will implement the requirement (FR7) and must also conform to the requirement (PR2).

1.1.5 Website

The website will consist of a set of web-pages which implement all the required functionalities for the user (FR8) and (FR9). It must also conform to the requirements (EIR1), (PR1) and (PR2). The website will communicate with the Web API via HTTP to receive from and send data to the database. The website will have no communication with the Android application. It will also not directly communicate with the database, but will go through the web API.

1.2 Significant Android Components

1.2.1 Significant UI classes

HomeActivity	This class will hold the code to allow a user to move on to the NewRecordingActivity via a startNewRecordingButton.
NewRecordingActivity	This class will hold the code to allow a user to enter information such as Name, Phone Number, E-Mail, and Site Location. It will also allow the application to receive date and time information from the Android device. The nextButton will then move the user to the AddNewSpeciesActivity.
AddNewSpeciesActivity	This class will hold the code to allow a user to enter all the required information about a species entry in the current recording. It allows you to choose the name of a species from a locally saved list, and also allows the user to add a new species name, if not found. This activity must have the functionality to use the device's GPS capabilities to record location information of the species. It allows the user to select abundance in accordance with the DAFOR scale. The user should be able to add a scene/specimen picture through the device's camera or the gallery application. The user can also add a note if they wish. There is then a confirmButton which adds the species to the current recording and moves the user on to the MaintainRecordingActivity.
MaintainRecordingActivity	This class will hold the code to allow a user to maintain the current recording. It will contain the functionality to edit any entered species from the collection in the recording. It will also allow the user to delete the current recording, removing all stored species data. The user will be able to save the recording, sending the current recording to the server at the first opportunity.

1.3 Significant other classes

- Specimen This class will hold the code to define all the information a specimen object can hold. Fields to hold this data will be: speciesName, gpsLocation, abundance, scenePicture, specimenPicture, notes. This class will also provide getter and setter methods for the mentioned fields.
- Recording This class will hold the code to define all the information a recording object will hold. Fields to hold this data will be: usersName, usersPhoneNumber, usersEmail, usersAddress, dateTime. This class will contain getters and setters for the mentioned fields.

1.4 Significant Website Components

- Navigation A set of buttons at the top of each webpage that will allow the user to move around the site. When clicking on the desired button, they will be taken to the designated page.
- Homepage This landing page will provide all the general information a user needs to know about the service. It also contains a search bar which will allow the user to filter through the plant database to find what they are looking for.
- View Species Page This page will display every species in the database for the user to view and select. If the user selects a species, they will be taken to the View Chosen Species Page, which provides more species information. This page will also allow the user to add a new species, taking them to the Add Species Page.
- View Chosen Species Page This page will display all information about the specific species chosen by the user.
- Add Species Page This page will provide all the forms necessary to input all data for a chosen species (name, location, abundance, scene picture, specimen pictures and notes), or allow a user to add a new species that is not listed.
- Map Overlay This component will be a pop-up map that will show the locations of the chosen species. This location will be taken from the database.

1.5 Significant Server Components

Server	A database and a set of Web API commands to allow manipulation.
Database	a MySQL database where we will store all data for our system
Web API	A list of commands that allow the Android and website systems to access and manipulate data in the Database.

- Tables
 - botany_users
 - botany_records
 - botany_specimens
 - botany_reserves
 - botany_resources
- addRecord.php - Adds a record to the database
- addReserve.php - Adds a reserve to the database
- addResource.php - Adds a resource (picture) to the database
- authenticateAdmin.php - Checks an inputted password and tells if it's the same as the stored password
- getRecord.php - Returns a record
- getRecords.php - Returns all records
- getReserve.php - Returns a reserve
- getReserves.php - Returns all reserves
- getResource.php - Returns a resource (picture)
- getSpecimen.php - Returns a specimen
- getSpecimens.php - Returns all specimens
- removeReserve.php - Removes a reserve
- removeSpecimen.php - Removes a specimen
- updateReserve.php - Updates a reserve
- updateSpecimen.php - Updates a specimen

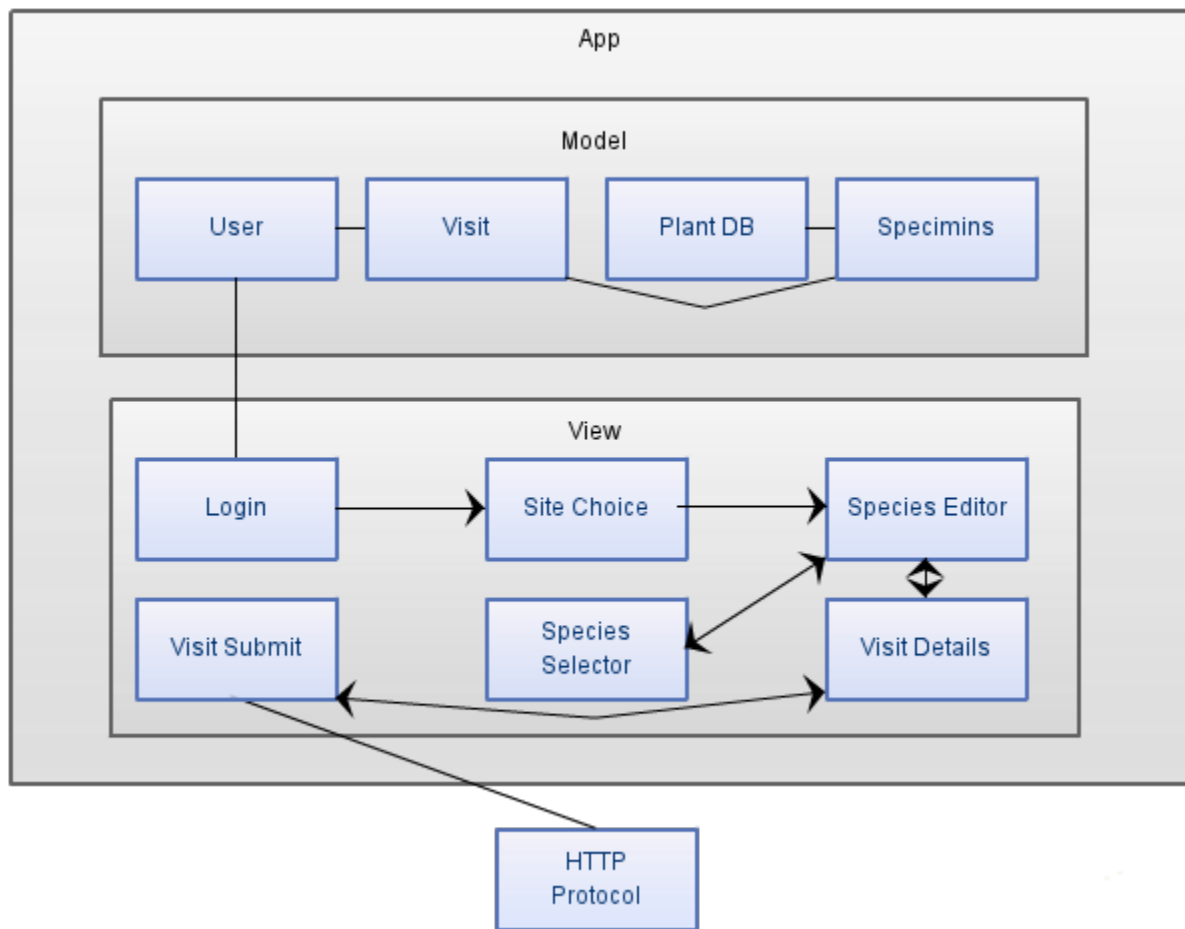


Figure 1: Component diagram for Android

2 Android Application Design

2.1 Decomposition

The compilation dependencies are as follows:

- Android dependencies a user will have an array of visits
- A visit will have an array of specimens
- A specimen can get data from the plant db (Fig 1).

2.2 Interfaces

2.2.1 Login Activity Interface

```

1  /**
   * A login screen that offers login via email and password.
   */
   public interface LoginPage extends Activity implements LoaderCallbacks<Cursor>
   {

6      /**
       * Called when building page for the GUI interface;
       */

       protected void onCreate(Bundle savedInstanceState);

11      /**
       * AutoFill
       */
       private void populateAutoComplete();

16      /**
       * Check email
       */
       private boolean isEmailValid(String email);

21      /**
       * Shows the progress UI and hides the login form.
       */
       @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
26      public void showProgress(final boolean show);

       /**
       * Called when building page for the GUI ;
       */
31      public Loader<Cursor> onCreateLoader(int i, Bundle bundle);

       /**
       * Called when building page for the GUI is finished;
       */
36      public void onLoadFinished(Loader<Cursor> cursorLoader, Cursor cursor);

       public void onLoaderReset(Loader<Cursor> cursorLoader);

41      /**
       * Called when building page for the GUI interface;
       */
       private void addEmailsToAutoComplete(List<String> emailAddressCollection);

46      /**
       * Represents an asynchronous login/registration task used to authenticate
       * the user.
       */
51      public interface UserLoginTask extends AsyncTask<Void, Void, Boolean> {

```

```
protected Boolean doInBackground(Void... params);  
protected void onPostExecute(final Boolean success);  
56 protected void onCancelled();  
    }  
}
```

android/src/loginInterface.java

2.2.2 Site Choice Activity Interface

```
public class SiteChoice extends Activity {  
4    /**  
    * Called when building page for the GUI interface;  
    */  
    protected void onCreate(Bundle savedInstanceState);  
  
9    /**  
    * Inflate the menu; this adds items to the action bar if it is present.  
    */  
    public boolean onCreateOptionsMenu(Menu menu);  
  
14    /**  
    * Handle action bar item clicks here. The action bar will  
    * automatically handle clicks on the Home/Up button, so long  
    * as you specify a parent activity in AndroidManifest.xml.  
    */  
19    public boolean onOptionsItemSelected(MenuItem item);  
}
```

android/src/siteChoiceInterface.java

2.2.3 Species Selector Activity Interface

```
public class SpeciesSelector extends Activity {  
3    /**  
    * Called when building page for the GUI interface;  
    */  
    protected void onCreate(Bundle savedInstanceState);  
8  
    /**  
    * Inflate the menu; this adds items to the action bar if it is present.  
    */  
    public boolean onCreateOptionsMenu(Menu menu);  
13  
    /**  
    * Handle action bar item clicks here. The action bar will  
    * automatically handle clicks on the Home/Up button, so long  
    * as you specify a parent activity in AndroidManifest.xml.  
18    */  
    public boolean onOptionsItemSelected(MenuItem item);  
}
```

android/src/speciesSelectorInterface.java

2.2.4 Visit Management Activity Interface

```
public class VisitManagement extends Activity {  
3    /**  
    * Called when building page for the GUI interface;  
    */  
    protected void onCreate(Bundle savedInstanceState);  
8  
    /**  
    * Inflate the menu; this adds items to the action bar if it is present.  
    */  
    public boolean onCreateOptionsMenu(Menu menu);  
13  
    /**  
    * Handle action bar item clicks here. The action bar will  
    * automatically handle clicks on the Home/Up button, so long  
    * as you specify a parent activity in AndroidManifest.xml.  
18    */  
    public boolean onOptionsItemSelected(MenuItem item);  
}
```

android/src/visitManagementInterface.java

2.2.5 Visit Submit Activity Interface

```
public class VisitSubmit extends Activity {  
3    /**  
    * Called when building page for the GUI interface;  
    */  
    protected void onCreate(Bundle savedInstanceState);  
8  
    /**  
    * Inflate the menu; this adds items to the action bar if it is present.  
    */  
    public boolean onCreateOptionsMenu(Menu menu);  
13  
    /**  
    * Handle action bar item clicks here. The action bar will  
    * automatically handle clicks on the Home/Up button, so long  
    * as you specify a parent activity in AndroidManifest.xml.  
18    */  
    public boolean onOptionsItemSelected(MenuItem item);  
}
```

android/src/visitSubmitInterface.java

2.2.6 User Data Class Interface

```
public class User{  
3    /**  
    * Returns the users first name  
    */  
    public String getUserForeName();  
  
8    /**  
    * Returns the users last name  
    */  
    public String getUserLastName();  
  
13   /**  
    * Returns the users phone number  
    * Formatted accordingly (Eg 07...)  
    */  
    public String getUserPhoneNumber();  
  
18   /**  
    * Returns the users email  
    */  
    public String getUserEmail();  
23 }  
  
    android/src/userClassInterface.java
```

2.2.7 Visit Data Class Interface

```
public class Visit{
2
    /**
    * Returns the visit Date when the user logged in
    * https://docs.oracle.com/javase/7/docs/api/java/sql/Date.html
    * Fulfils requirement FR2
7    */
    public Date getVisitDate();

    /**
12    * Returns the visit Time when the user logged in
    * https://docs.oracle.com/javase/7/docs/api/java/sql/Time.html
    * Fulfils requirement FR2
    */
17    public Time getVisitTime();

    /**
    * Returns a list of all specimens entered so far
    * Fulfils FR3
22    */
    public Specimen[] getSpecimen();

    /**
    * Adds a specimens to the visit
    * Fulfils FR3
27    */
    public void addSpecimen(Specimen s);
}
```

android/src/visitClassInterface.java

2.2.8 Plant DB Interface

This Will be a file obtained from the Botanical Society of Britain and Ireland containing a large list of known plant types in a comma sperated list (CSV) format. This file will be parsed at runtime and a condensed list created in alphabetical order for searching. Obtained from <http://www.bsbi.org.uk/resources.html>

2.2.9 Specimen Class Interface

```

public class Specimen{

    /**
4      * Abundance scale
      * Fulfills FR4
      */
    public Enum AbundanceEnum {
        DOMINANT,
9        ABUNDANT,
        FREQUENT,
        OCCASIONAL,
        RARE,
14    }

    /**
14      * Gets the approximate specimen longitude
      */
    public String getLatitude();

19    /**
      * Gets the approximate specimen latitude
      */
    public String getLongitude();

24    /**
      * Gets the abundance rating
      */
    public AbundanceEnum getAbundance();

29    /**
      * returns the free text comment made by the user
      */
    public String getComment();

34    /**
      * returns a URI to the scene photo if one is provided
      */
    public String getScenePhotoURI();

39    /**
      * returns a URI to the specimen photo if one is provided
      */
    public String getSpecimenPhotoURI();

44    /**
      * takes a specimen photo.
      * Opens up the camera and allows the user to take a photo rather than

```

```

    * asking for the location of one
    */
49 public void takeSpecimenPhoto();

    /**
    * takes a specimen photo
54 * Opens up the camera and allows the user to take a photo rather than
    * asking for the location of one
    */
    public void takeScenePhoto();
}

```

android/src/SpecimenClassInterface.java

3 Web Design

3.1 Interface

These are the server side scripts that will be used across the website.

Constants

- Header
 - session_start();
 - Doctype
 - meta tags
 - * description
 - * Keywords
 - * authir
 - * content type
 - * CSS links
 - * Font link
 - * validation.js
 - * \$title
 - Header/Div
 - * Logo
- Nav
 - Nav
 - include‘breadcrumbs’;
- Footer
 - Site map link
 - Icon
- config.php
 - Address of Server

- Admin Password
- Index
 - \$title
 - include 'header.php';
 - include 'nav.php';
 - text
 - include 'simple_search.php';
 - text
 - * app link
 - include 'footer.php';
- plants
 - include 'config.php';
 - \$title
 - include 'header.php';
 - include 'nav.php';
 - add_plant_record link
 - include 'advanced_search.php';
 - include 'plant_db_declare.php';
 - include 'plant_sorting.php';
 - include 'footer.php';
- add_plants_record
 - include 'config.php';
 - \$title
 - include 'header.php';
 - include 'nav.php';
 - form(input)
 - if no image uploaded, default image else uploaded image
 - include 'upload_image.php';
 - include 'save_record.php';
 - a href 'plants.php' cancel
 - include 'footer.php';
- plants_record
 - include 'config.php';
 - \$title

- include ‘header.php’;
 - include ‘nav.php’;
 - \$specific record details
 - a href ‘edit_plant_record.php’ edit
 - a href ‘delete_plant_record.php’ remove
 - * delete prompt js
 - individual record photo or default image
 - a href ‘plants.php’ where to find
 - * (plant_map.js)
 - include ‘footer.php’;
- edit_record
 - include ‘config.php’;
 - \$title
 - include ‘header.php’;
 - include ‘nav.php’;
 - include ‘breadcrumbs.php’;
 - \$individual record
 - a href ‘edit_plant_record.php’ edit
 - a href ‘delete_plant_record.php’ remove
 - \$individual record photo
 - a href ‘plants.php’ where to find
 - * (plant_map.js)
 - include ‘footer.php’;

3.2 Detailed design

3.2.1 config.php

config will open a link to the mysql database at the start of each new session, all changes to records and searches are reliant on this script.

```
$conn = mysql_connect(host=<localhost>  
port=3306 dbname=<nameofdatabase>  
user=<uid> password=<password>);
```

The script will also include a few lines closing the link to the database at the end of the session as not to produce more connections to the database than needed.

3.2.2 plant_sorting.php

The plant data can be sorted in multiple different ways there will be a default sort using SORT_REGULAR however the options of SORT_STRING and SORT_NUMERIC will also be available as required. The script will provide the user with an option of how they wish to sort the data.

3.2.3 simple_search.php

The simple search script will search the database for a string matching the users input, all entries in the database containing that string will be returned. It will require a connection to the database from config.php, if there is no connection then the script will return an error message to notify the user of a problem with the connection to the database.

3.2.4 advanced_search.php

The script for advanced search will allow for searches by string, ID, data type, location and user. The user should select their search type before inputting, the results shall be returned unless no result is found or there is an issue with the connection to the database in which case an error message shall be returned to the user.

3.2.5 edit_plant_record.php

This script will take data input from the client side to form an update in an object oriented way. The update will set the field to be updated to the new data. The script will query if the connection to the server is valid before committing the changes, it will also return a confirmation to the user that the record has been updated or that there was a problem connecting to the server (they have timed out) so they can try again.

3.2.6 delete_plant_record.php

The script for deleting a record will be very straight forward, using an object oriented approach the script will request for the record to be deleted by using its ID. Much like the edit this script will return a confirmation to the user that the record has successfully been deleted or that there was a problem with the connection to the server.

3.2.7 upload_image.php

This script should prompt the user to choose an image file to upload first, when an image file has been selected an upload function will collect the data about the file where it can be validated to check it matches the requirements we have set (eg: file type and size). The image file, if compatible, will be uploaded and the script will return a success message, otherwise a message will be returned to the user explaining which parameter is incorrect or if there is an issue with the connection to the server.

3.2.8 regex validation

validation will be done on the client side using javascript and regular expressions. Doing this allows us to use an onchange method when the user is inputting data with fixed parameters, such as what characters can be used, without putting extra load on the server.

3.2.9 Index.php

The index script will be the home page for the web site, allowing users to read about the Botany Project, navigate the web site, search for a plant and download the app. There will be a header at the top of the page containing the Botany Group Project title and an image of

the logo. The banner will be customised using CSS. Underneath there will be a navigation bar using the nav.php script. Below this there will be two text boxes which will be achieved using the echo function and will be customised to have a white background using CSS. Between the texts there will be a search bar controlled by simple_search.php. At the bottom the footer.php will be included. A background image will be used for the homepage.

3.2.10 Header.php

The header will be responsible for creating a session for the user. The header script will be responsible used to declare the doctype and meta tags such as description, keywords, content type, CSS and font links, validation.js and title.

3.2.11 footer.php

The footer will contain the site map link allowing users to access all the pages of the website. There will also be an image of the logo in the centre of footer.

3.2.12 nav.php

The navigation bar provides an easy reference for the contents of the web site and enables the user to navigate the web site conveniently. This will be achieved by attaching a div class to the navigation text with a href attribute, linking it to the corresponding pages. CSS will be used to style the navigation bar and links accordingly with the Web user interface design. Underneath there will be breadcrumbs included using the breadcrumbs script.

3.2.13 Edit/add record

Upon selecting an individual plant from the database, the plant along with its attributes and an image will be displayed by pulling it from the database using SQL. The user can find where to locate the plant by clicking the Where to find button which will run the plant_map.js script, loading Google maps javascript API. The user can then edit the plants information by clicking the edit button supplied, running the edit_plant_record.php script, loading the Edit Record page. This page will use the config.php script to form a database connection that will insert a table, ready to retrieve and print the selected fields of the plant chosen by the user. This will be achieved using pg_query; selecting the required fields and echoing a table. A while loop will be used to read and pull data from the database. Users can then edit plant information by filling in the form with the attributes assigned to a plant. The form is to be validated using javascript, identifying the required fields. If a required field is left blank, an alert will pop up requesting the user to fill the field in. The image can also be updated by clicking the supplied Upload Image submit button, running the upload_image.php script. Once the form has been edited by the user, they can then save the record using the Save button linked to the save_record.php script. This will update the database with the new edited record. Users will also be given the option to add new plant records to the database. This page will be the same as the Edit Record page but with a blank form for the user to fill in.

3.2.14 delete record

Users are given the option to delete a specific plant record. A remove button linked to the delete_plant_record.php script will be supplied to do this.

3.2.15 delete confirmation pop up

If a user clicks the remove button, a javascript alert pop up will be displayed confirming the deletion of a record. This has been included to avoid accidental deletions.

3.2.16 config.php

The config.php file will be used to establish the connection to the mysql database using the servers API at the start of a new session.

```
$CONFIG=array  
api="file location"  
session="where the session is stored"
```

The config.php will need to be included in all php scripts.

3.2.17 authenticate.php

The authenticate.php script will be used to authenticate an admin user on the site. Who through entering a correct password will have access to certain features of the site that regular users will not have. Such as deletion of reserves and specimens. The authentication will be checked to the servers API. If a correct password is entered then the user will see a message confirming it. If an incorrect password is used then a different message will be seen saying that a wrong password has been used.

3.2.18 delete_curl.php

This script is used to delete specific specimens from the servers database. This feature will only become available once a user has entered a correct password to show that they are an admin user. PHP CURL is used so that the website can communicate with the server.

3.2.19 edit_specimens.php

This script is similar to the delete script. It uses PHP CURL to communicate with the server. This function is also only available to admin users.

3.2.20 footer.php

This script will be included on every page. It will contain a small website logo in the centre of the footer.

3.2.21 get_reseve.php

This will be used to call all the data about the reserves stored on the server. PHP CURL will be used to communicate with the server and to decode the JSON that the API will be using so that it can be displayed on the reserves page.

3.2.22 header.php

The header will be responsible for creating a session and storing it for the users. The header script will be responsible for declaring the doctype and meta tags such as description, keywords, content type, CSS and font links, validation as well as JAVASCRIPT and the title.

3.2.23 img_curl.php

This is used to find get the specimen and scene photo from the database. PHP CURL is again used to communicate with th server.

3.2.24 logout.php

This script is used to logout an admin user who has entered a correct password to the site. It will end the session that is being used and display a message telling the user that they have been logout of the site.

3.2.25 map.php

The contains the JAVASCRPIT to be able to view the plants location within a Google maps pop up. It will use the latitude and longitude variables of each specimen to give an accurate location.

3.2.26 specimens_curl.php

This will be used to access the information and each individual specimen. CURL is again used to access the server.

3.2.27 resdelete_curl.php

This will work in a very similar way to the delete_curl.php script but will be used for deleting whole reserves instead of just individual specimens. The user will need the be logged in for this function to work.

3.2.28 reserves_curl.php

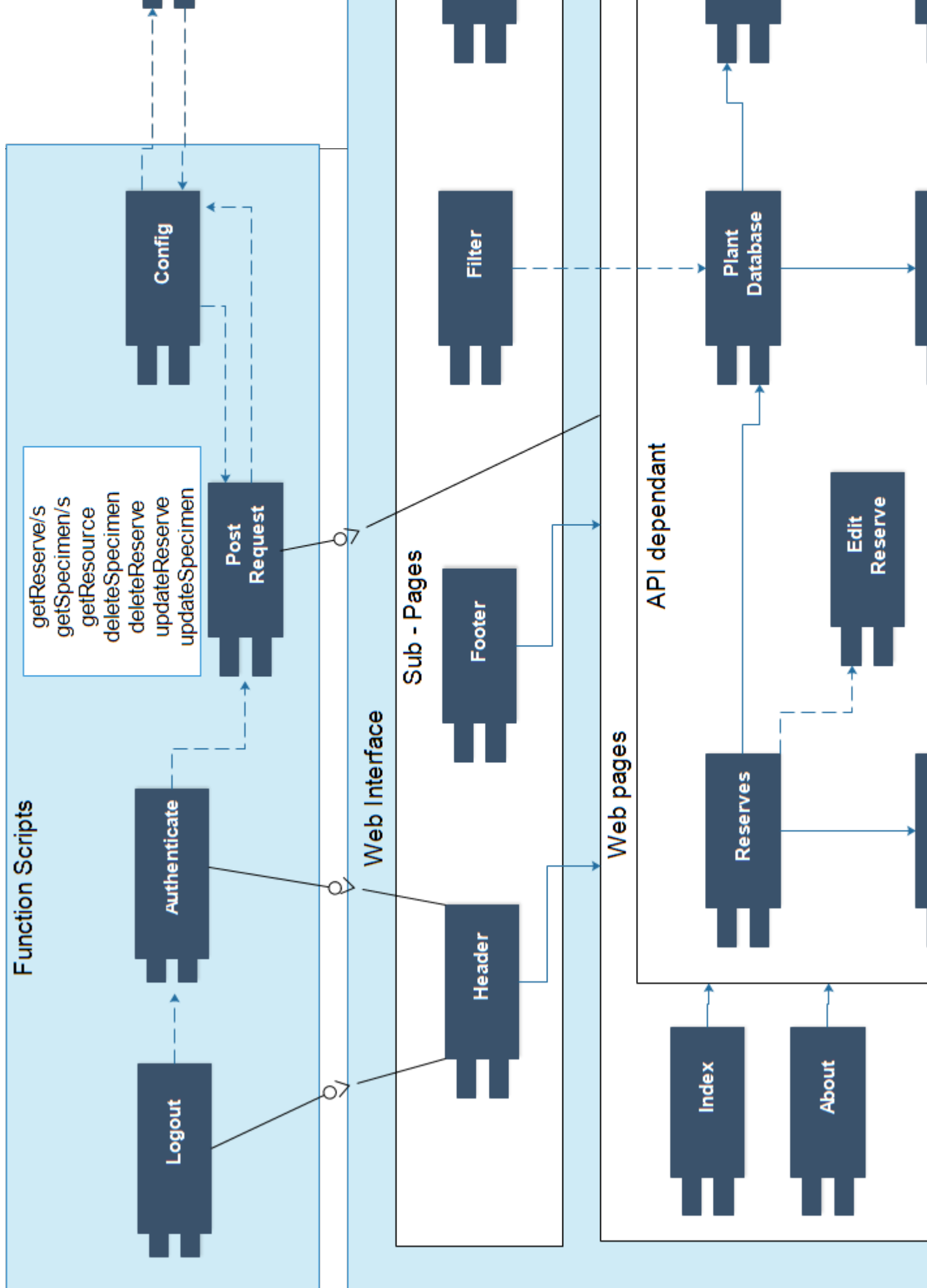
Much like the record_curl.php it will be used to get information form the server, but information about the reserves rather then the specimen details.

3.2.29 specimens_search.php

This will be used to search for certain specimens within the specimen page on the website. Users will be able to search by Species Name, Location Name and The User Name of people using the site.

3.2.30 filter.php

This will also be used on the plants page on the website and will allow users to order by Species Name, Location, User Name, Date and Abundance and allow results to be show in ascending or descending order.



4 Server Design

4.1 Interface

These are the commands that the web API will be using to maintain the database and receive commands.

4.1.1 AuthenticateAdmin

Checks the inputted password against the listed valid password to allow website use.

- Headers required:
 - none
- POST arguments:
 - password : String
- Statuses:
 - 200 OK:
 - * boolean : true/false
 - 400 Bad Request
 - 500 Internal Server Error

4.1.2 AddRecord

Adds a record to the database, and returns the record ID. Used by the Android application.

- Headers required:
 - none
- POST arguments:
 - record : Record
- Statuses:
 - 200 OK:
 - * record ID : Integer
 - 400 Bad Request
 - 500 Internal Server Error

4.1.3 AddReserve

Adds a reserve to the database, and returns the reserve ID.

- Headers required:
 - none
- POST arguments:
 - reserve : Reserve
- Statuses:
 - 200 OK:
 - * reserve ID : Integer
 - 400 Bad Request
 - 500 Internal Server Error

4.1.4 RemoveSpecimen

Removes a single specimen from the database, provided an ID and an administrator password. Used by the web site.

- Headers required:
 - none
- POST Arguments:
 - specimenID : Integer
 - password : String
- Statuses:
 - 200 OK : no data
 - 400 Bad Request
 - 401 Unauthorized
 - 500 Internal Server Error

4.1.5 GetSpecimen

Returns a single specimen result, which includes data from the record the specimen belongs to when provided an ID. Used by the web site.

- Headers required:
 - none
- POST Arguments:
 - specimenID : Integer

- Statuses:
 - 200 OK :
 - * specimen : SpecimenResult
 - 400 Bad Request
 - 500 Internal Server Error

4.1.6 UpdateSpecimen

Updates a specific Specimen by ID.

- Headers Required:
 - none
- POST Arguments:
 - specimenID : int
 - password : String
- Statuses:
 - 200 OK :
 - 400 Bad Request
 - 500 Internal Server Error

GetSpecimens Returns a list of specimen results, which include data from the record the specimen belongs to, searching and sorting them by defined criteria. Used by the web site.
- Headers Required:
 - none
- POST Arguments:
 - order : String (“ascending” or “descending”)
 - method : String (“speciesName”, “locationName”, “userName”, “timestamp”)
 - value : String
 - column : String (“speciesName”, “locationName”, “userName”)
- Statuses:
 - 200 OK : array of SpecimenResult
 - 400 Bad Request
 - 500 Internal Server Error

4.1.7 AddResource

Adds a file resource to the server, and returns an ID. Used by the Android application and the web site.

- Headers required:
 - none
- POST Arguments:
 - resource : OctetStream
- Statuses:
 - 200 OK :
 - * resource ID : Integer
 - 400 Bad Request
 - 500 Internal Server Error

4.1.8 GetResource

Returns a file resource when provided with a resource ID. Used by the web site.

- Headers required:
 - none
- POST Arguments:
 - resourceID : Integer
- Statuses:
 - 200 OK :
 - * resource data : OctetStream
 - 400 Bad Request
 - 500 Internal Server Error

4.1.9 GetReserve

Returns a single reserve result, which holds data about a location used by records.

- Headers required:
 - none
- POST Arguments:
 - reserveID : Integer
- Statuses:

- 200 OK :
 - * reserve : Reserve
- 400 Bad Request
- 500 Internal Server Error

4.1.10 GetReserves

Returns a list of distinct locations used by records in the database for use by the Android application.

- Headers Required:
 - none
- POST Arguments:
 - none
- Statuses:
 - 200 OK : array of Reserve
 - 400 Bad Request
 - 500 Internal Server Error

4.1.11 UpdateReserve

Updates a specific Reserve by ID.

- Headers Required:
 - none
- POST Arguments:
 - reserveID : int
 - password : String
- Statuses:
 - 200 OK :
 - 400 Bad Request
 - 500 Internal Server Error

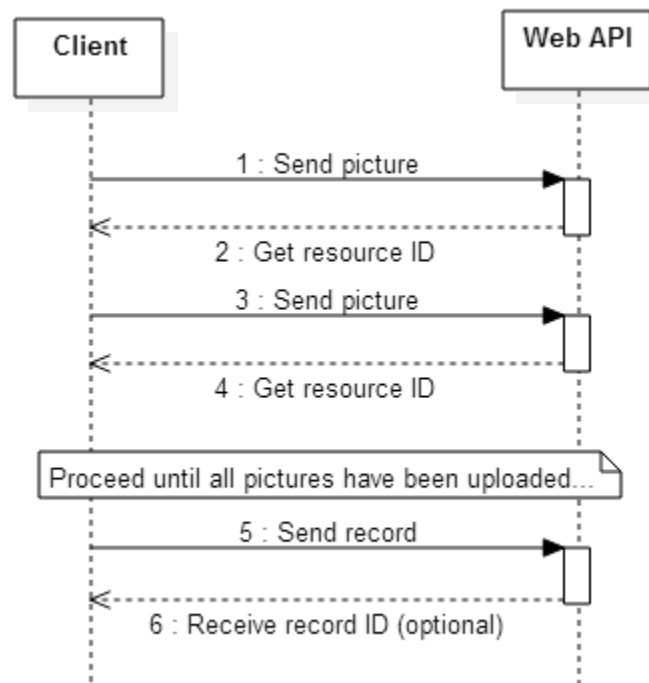


Figure 3: Sequence diagram: adding a record through the web API

4.1.12 RemoveReserve

Removes a specific Reserve by ID.

- Headers Required:
 - none
- POST Arguments:
 - reserveID : int
 - password : String
- Statuses:
 - 200 OK :
 - 400 Bad Request
 - 500 Internal Server Error

4.2 Detailed design

4.2.1 Diagrams

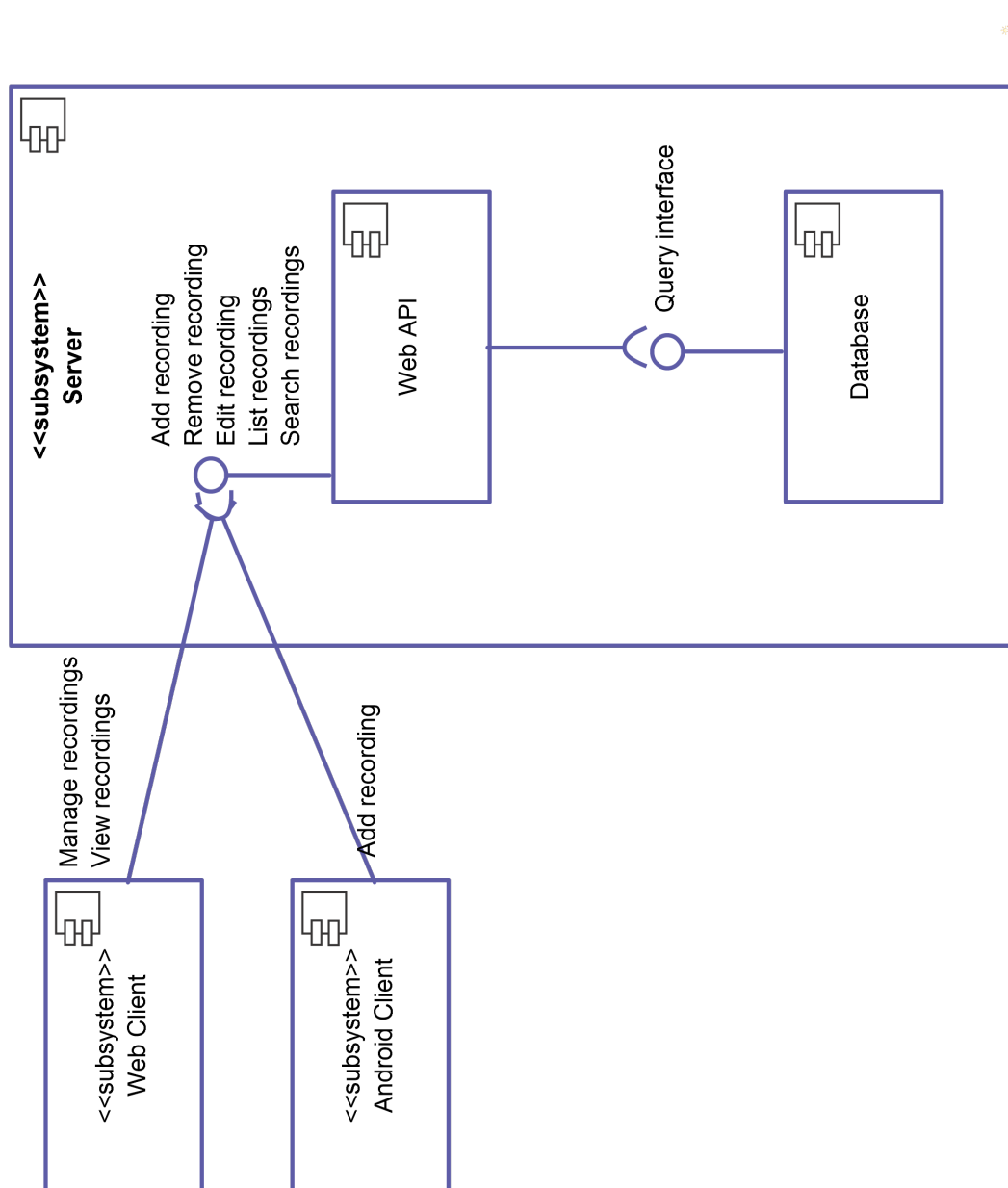


Figure 4: Web Api component Diagram

4.2.2 Significant data structures

The web API is going to make use of the Record, Specimen, RecordLocation and SpecimenResult data structures, which will be exchanged between the server and its clients (the Android application and the website). They will be represented in a JSON format, which is readily available for use in PHP, JavaScript, and Android. Data types, where specified, are JSON data types.

The structure of a Record is as follows:

- Record : Object
 - UserName : String
 - UserPhone : String
 - UserEmail : String
 - LocationName : String
 - Timestamp : Number
 - LocationOS : String
 - Specimens : Array of Specimen

The structure of a Reserve is as follows:

- Reserve : Object
 - ReserveID : Number
 - LocationName : String
 - LocationOS : String
 - Description : String

The structure of a Specimen is as follows:

- Specimen : Object
 - SpeciesName : String
 - LocationLatitude : Number
 - LocationLongitude : Number
 - Abundance : Number
 - Comment : String
 - ScenePhoto : String (ID of a resource on the server)
 - SpecimenPhoto : String (ID of a resource on the server)

The structure of a SpecimenResult is as follows:

- SpecimenResult : Object
 - UserID : Number
 - RecordID : Number
 - SpecimenID : Number

- UserName : String
- UserEmail : String
- UserPhone : String
- LocationName : String
- LocationOS : String
- Timestamp : Number
- SpeciesName : String
- LocationLatitude : Number
- LocationLongitude : Number
- Abundance : Number
- Comment : String
- ScenePhoto : String (ID of a resource on the server)
- SpecimenPhoto : String (ID of a resource on the server)

The web API uses config.php as a configuration store. It contains the \$CONFIG array, containing the following entries:

- dbname : the address of the MySQL database to be used
- username : the user name to use on the MySQL database
- password : the password to use on the MySQL database
- adminPassword : the password used in the DeleteSpecimen web method

The web API is going to use a relational database as a data store. The database tables are as follows:

- Users
 - UserId : INT auto-increment PK
 - UserName : VARCHAR(20) not-null
 - UserFullName : VARCHAR(50)
 - UserPhone : VARCHAR(20)
 - UserEmail : VARCHAR(50)
 - UserPassword : BINARY(20)
- Records
 - RecordId : INT auto-increment PK
 - UserId : INT not-null
 - LocationName : VARCHAR(50)
 - Timestamp : INT
 - LocationOS : VARCHAR(10)

- Resources
 - ResourceId : INT auto-increment PK
- Reserves
 - ReserveID : INT auto-increment PK
 - LocationName : VARCHAR(50)
 - LocationOS : VARCHAR(10)
 - Description : TEXT
- Specimens
 - SpecimenId : INT auto-increment PK
 - RecordId : INT not-null
 - SpeciesName : VARCHAR(255) not-null
 - Latitude : FLOAT(10,6)
 - Longitude : FLOAT(10,6)
 - Abundance : INT
 - Comment : TEXT
 - ScenePhoto : INT
 - SpecimenPhoto : INT

References

- [1] Project Plan Specification Standards :B. P. Tiddeman (2014-09-23) SE.QA.05b 1.2

5 Document History

Version	Edit	Date	Persons
0.1	Initial Version	November 5 2014	nid21
0.2	Updated with decomposition description	November 12 2014	nid21
0.3	Updated with Android interfaces	November 17 2014	nid21
0.4	Included web and server sections	November 27 2014	nid21
0.5	Document review	November 28 2014	nid21
0.6	More document reviewing and preparing for re-lease	December 02 2014	nid21