# 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 | January 21, 2015 |
| Reference | SE_O2_DS_00 |



| | |
|---|---|
| Department | Computer Science |
| Address | Aberystwyth University |
| | Penglais Campas |
| | Ceredigion |
| | SY23 3DB |

# Contents

# 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 AddNewSpeciesActivty. |
| 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 Web API Components

AddRecord — Add a record received via a HTTP POST request in JSON format, checking it for validity, and return an appropriate status message as follows:

- 200 OK: The record has been saved to the database successfully;
  - Contains the record ID.
- 400 Bad Request: The method has been provided with malformed data.
- 500 Internal Server Error: The database is inaccessible or an error has occurred in the method itself.

RemoveRecord — Remove a record the database by an ID specified in a HTTP POST request, together with an Authorization header as specified in the HTTP protocol, carrying administrator credentials, and return an appropriate status message as follows:

- 200 OK: The record has been deleted from the database successfully.
- 400 Bad Request: Bad record ID specified.
- 401 Unauthorized: The method has been provided no or wrong authorization header.
- 500 Internal Server Error: The database is inaccessible or an error has occurred in the method itself.

ModifyRecord — Modify a record in database by an ID specified in a HTTP POST request, together with an Authorization header carrying administrator credentials as specified in the HTTP protocol, and the record data in a JSON format, returning an appropriate status message as follows:

- 200 OK: The record has been modified successfully.
- 400 Bad Request: Bad record ID or bad record data specified.
- 401 Unauthorized: The method has been provided no or wrong authorization header.
- 500 Internal Server Error: The database is inaccessible or an error has occurred in the method itself.

GetRecord — Retrieve a record from database by an ID specified in a HTTP POST request. and return an appropriate status message as follows:

- 200 OK: The record has been retrieved from the database successfully;
  - Contains the requested record in JSON.
- 400 Bad Request: Bad record ID specified.
- 500 Internal Server Error: The database is inaccessible or an error has occurred in the method itself.

GetRecords — Retrieve a number of records from database. Optional arguments such as the number/range of desired results, and filtering based on record properties can be specified in the POST request. Return an appropriate status message as follows:

- 200 OK: The records has been retrieved from the database successfully;
  - Contains the list of records in JSON format. An empty list is a possibility.
- 400 Bad Request: Bad arguments specified.
- 500 Internal Server Error: The database is inaccessible or an error has occurred in the method itself.

AddResource — Uploads a file resource to the server via POST, checking it for validity. Content-Type is to be multipart/form-data. Return an appropriate status message as follows:

- 200 OK: The resource has been saved successfully;
  - Contains the resource ID.
- 400 Bad Request: The method has been provided with malformed data.
- 500 Internal Server Error: The resource is inaccessible or an error has occurred in the method itself.

GetResource — Downloads a file resource from the server via POST, by specifying a resource ID. Return an appropriate status message as follows:

- 200 OK The resource has been found;
  - Contains the resource, with Content-Type: application/octet-stream.
- 400 Bad Request: The method has been provided with a bad ID.
- 500 Internal Server Error: The resource is inaccessible or an error has occurred in the method itself.

# 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).

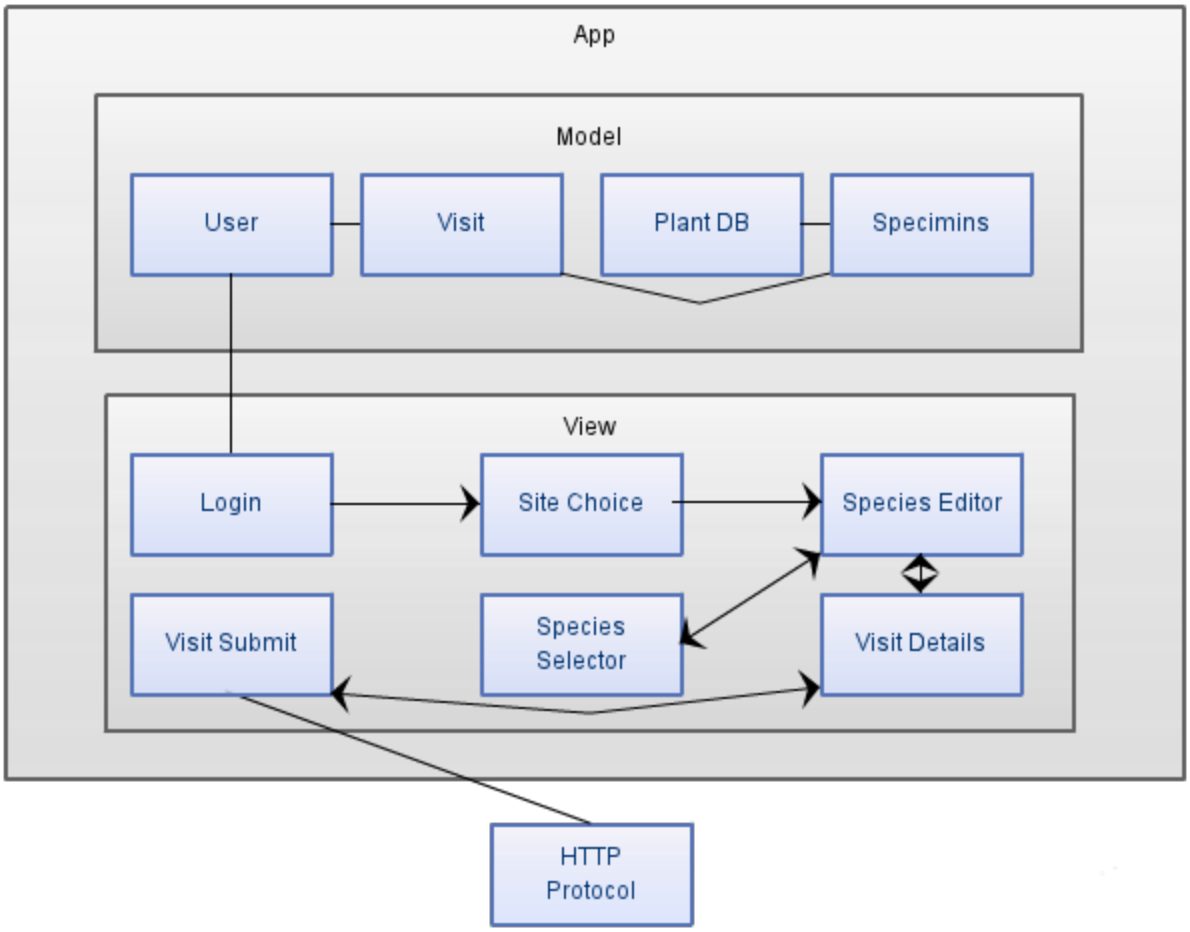



Figure 1: Component diagram for Android

## 2.2 Interfaces

### 2.2.1 Login Activity Interface

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

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

    protected void onCreate(Bundle savedInstanceState);

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

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

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


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

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


    public void onLoaderReset(Loader<Cursor> cursorLoader);

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

    /**
     * Represents an asynchronous login/registration task used to authenticate
     * the user.
     */
    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 {

    /**
     * Called when building page for the GUI interface;
     */
    protected void onCreate(Bundle savedInstanceState);

    /**
     * Inflate the menu; this adds items to the action bar if it is present.
     */
    public boolean onCreateOptionsMenu(Menu menu);

    /**
     * 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.
     */
    public boolean onOptionsItemSelected(MenuItem item);
}
```

android/src/siteChoiceInterface.java

### 2.2.3 Species Selector Activity Interface

```
public class SpeciesSelector extends Activity {

    /**
     * Called when building page for the GUI interface;
     */
     protected void onCreate(Bundle savedInstanceState);

     /**
     * Inflate the menu; this adds items to the action bar if it is present.
     */
     public boolean onCreateOptionsMenu(Menu menu);

     /**
      * 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.
      */
     public boolean onOptionsItemSelected(MenuItem item);
}
```

android/src/speciesSelectorInterface.java

### 2.2.4 Visit Management Activity Interface

```
public class VisitManagement extends Activity {

    /**
     * Called when building page for the GUI interface;
     */
    protected void onCreate(Bundle savedInstanceState);

    /**
     * Inflate the menu; this adds items to the action bar if it is present.
     */
    public boolean onCreateOptionsMenu(Menu menu);

    /**
     * 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.
     */
    public boolean onOptionsItemSelected(MenuItem item);
}
```

android/src/visitManagementInterface.java

### 2.2.5 Visit Submit Activity Interface

```
public class VisitSubmit extends Activity {

    /**
     * Called when building page for the GUI interface;
     */
    protected void onCreate(Bundle savedInstanceState);

    /**
     * Inflate the menu; this adds items to the action bar if it is present.
     */
    public boolean onCreateOptionsMenu(Menu menu);

    /**
     * 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.
     */
    public boolean onOptionsItemSelected(MenuItem item);
}
```

android/src/visitSubmitInterface.java

### 2.2.6 User Data Class Interface

```
public class User{

    /**
     * Returns the users first name
     */
    public String getUserForeName();

    /**
     * Returns the users last name
     */
    public String getUserLastName();

    /**
     * Returns the users phone number
     * Formatted accordingly (Eg 07...)
     */
    public String getUserPhoneNumber();

    /**
     * Returns the users email
     */
    public String getUserEmail();
}
```

android/src/userClassInterface.java

### 2.2.7 Visit Data Class Interface

```
public class Visit{

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


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


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

    /**
     * Adds a specimens to the visit
     * Fulfils FR3
     */
    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{

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

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

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

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

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

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

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

    /**
     * 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 6

## 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 timeout.php

Time out is essentially a script that will unset a specific session when no activity has taken place, display and alert message, which once confirmed will send the user back to the plant list page. This will mainly be included when adding or editing records, as if the computer is left alone, then anyone could distort the record information.

### 3.2.2 plants.php

The plants page is where the list of plants are going to be displayed. The page will require the connection to the database through the config file, and it must include the html pages header.php, nav.php and the footer.php files. The page will also have its own heading that will be created by a variable, will be declared above the header.php include, where the header will have $title so it can be adjusted easily. The main content area below the navigation, which is then split into two. The first section is where the user will be able to sort the

database records to their specific criteria. This sorting script will be placed in a separate file. The other half, is the main part of the page. In this section, the user can select a link that will allow them to create a new plant record and store it in the database. Under that, there is an advanced search, which will display records that comes under the criteria of their search input. This will also be in a separate script. This search will allow the user to not have to scroll through the long list of records. The plants are split into their scientific name and their common name. Once they are clicked, then they are taken to the plants individual record. The database itself is going to be called and declared from a separate script, where it will then place them into a table, which will present the plant lists.

### 3.2.3 db_declare.php

This script is basically where and how the plant list will be created and how it will be represented. Firstly, the table will have to be created, which will have a div attached to it. Then the scientific and common headings will be created. Once this has been created, an SQL statement will be used, making a handle onto the database connection, where it will select all the fields by using. We can then use the handle from the SQL statement to write out the database, using `while($row = pg_fetch_array ($res))` and then defining the cells by calling them by `$row['dbcolumn']` inside of td tags. Each of these rows will be links to the page that has their information on. The table is then closed. The script does require config.php, but this is already included in plants.php

### 3.2.4 Plant location

When showing the specific location of the plant, we will be using the google maps JavaScript API. To use this we will have to gain an API key from google;once we have access, we must activate the Google Maps API v3. To get the API key, we must give it some information about the website, such as the web address. Because we will be testing it on another server, we will register it to that site first, which we can change to the server we will be using it on later on at the final part of the project.

Implementing the code must start with giving the script source, included into the HTML head, where the API key must be inserted. Then a new JavaScript script will be created, stating an initialize function. This will be made up of a variable mapProp, where it will specify all of the options for the map, such as where to center the map, how close are we going to zoom in on the location and what type of map is it going to be, such as Roadmap and Terrain, which we will be using HYBRID, as it allows to see a combination of the roads and the scenery, which may be useful to a botanist trying to find the specific plant. In these options however, is the lat/lang input, which will locate where the plant is. Because this depends on the location of the plant that will be displayed, we will have to use a variable that will get the latitude and longitude from the record and place it in the variable so it can there be used to display its location. After these options have been defined, the mapProp variable is closed, and the map is then created through creating a new map variable, which will assign itself to a div depending on its ID. The script must then be closed. To display this, in the body, there must be a div with the same ID stated in the script.

### 3.2.5 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.

### 3.2.6 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.

### 3.2.7 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.

### 3.2.8 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.

### 3.2.9 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.

### 3.2.10 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.

#### 3.2.11 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.

#### 3.2.12 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.

#### 3.2.13 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.

#### 3.2.14 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.

#### 3.2.15 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.

#### 3.2.16 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.

#### 3.2.17 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

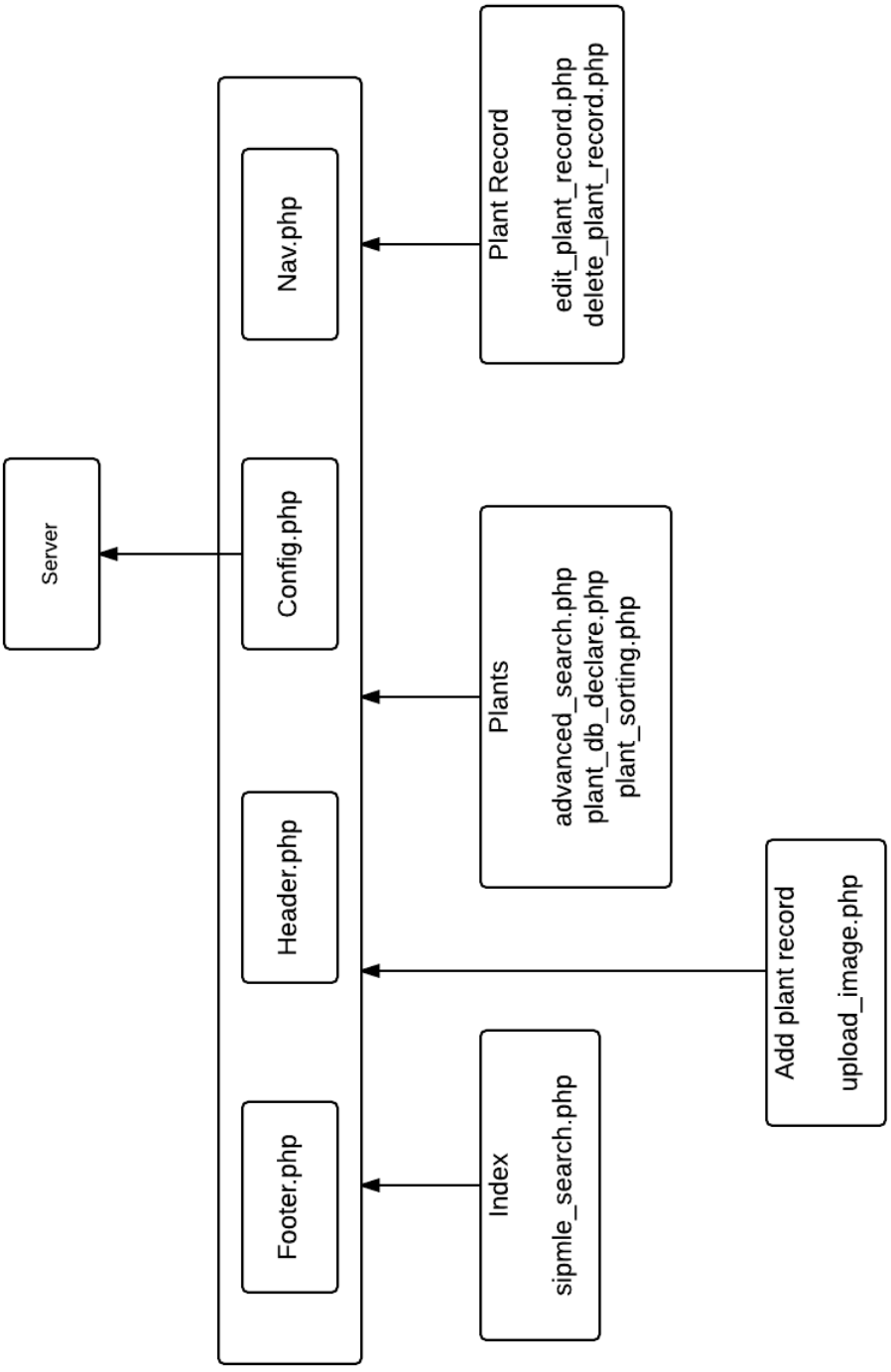
to locate the plant by clicking the Where to find button which will run the plant_map.js script, loading Google maps javascript API. The user can then edit the plants information by clicking the edit button supplied, running the edit_plant_record.php script, loading the Edit Record page. This page will 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.

#### 3.2.18 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.

#### 3.2.19 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.


Server
Config.php
Nav.php
Header.php
Footer.php
Plant Record
edit_plant_record.php
delete_plant_record.php
Plants
advanced_search.php
plant_db_declare.php
plant_sorting.php
Add plant record
upload_image.php
Index
sipmle_search.php

# 4 Server Design

## 4.1 Interface

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

- AddRecord
- Headers required:
  - none
- POST arguments:
  - record : Record
- Statuses:
  - 200 OK:
    - record ID : Integer
  - 400 Bad Request
  - 500 Internal Server Error
- RemoveRecord
- Headers required:
  - Authorization
- POST Arguments:
  - recordID : Integer
- Statuses:
  - 200 OK : no data
  - 400 Bad Request
  - 401 Unauthorized
  - 500 Internal Server Error
- ModifyRecord
- Headers required:
  - Authorization
- POST Arguments:
  - recordID : Integer
  - record : Record

- Statuses:
  - 200 OK : no data
  - 400 Bad Request
  - 401 Unauthorized
  - 500 Internal Server Error
- GetRecord
- Headers required:
  - none
- POST Arguments:
  - recordID : Integer
- Statuses:
  - 200 OK :
    - record : Record
  - 400 Bad Request
  - 500 Internal Server Error
- GetRecords
- Headers Required:
  - none
- POST Arguments:
  - maxResultCount : Integer
  - resultStartOffset : Integer?
  - speciesFilter : String?
  - abundanceFilter : Integer?
  - userNameFilter : String?
  - fullNameFilter : String?
  - commentFilter : String?
  - locationNameFilter : String?
  - locationLatitudeProximityFilter : Float?
  - locationLongitudeProximityFilter : Float?
  - locationProximityRadius : Float?
- Statuses:
  - 200 OK : array of Record

  - 400 Bad Request
  - 500 Internal Server Error

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

## 4.2 Detailed design

### 4.2.1 Diagrams

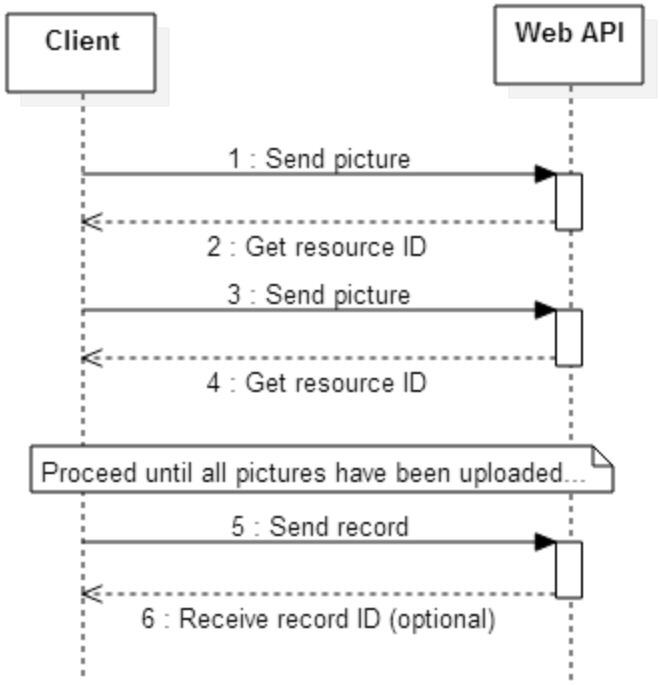



Figure 3: Sequence diagram: adding a record through the web API

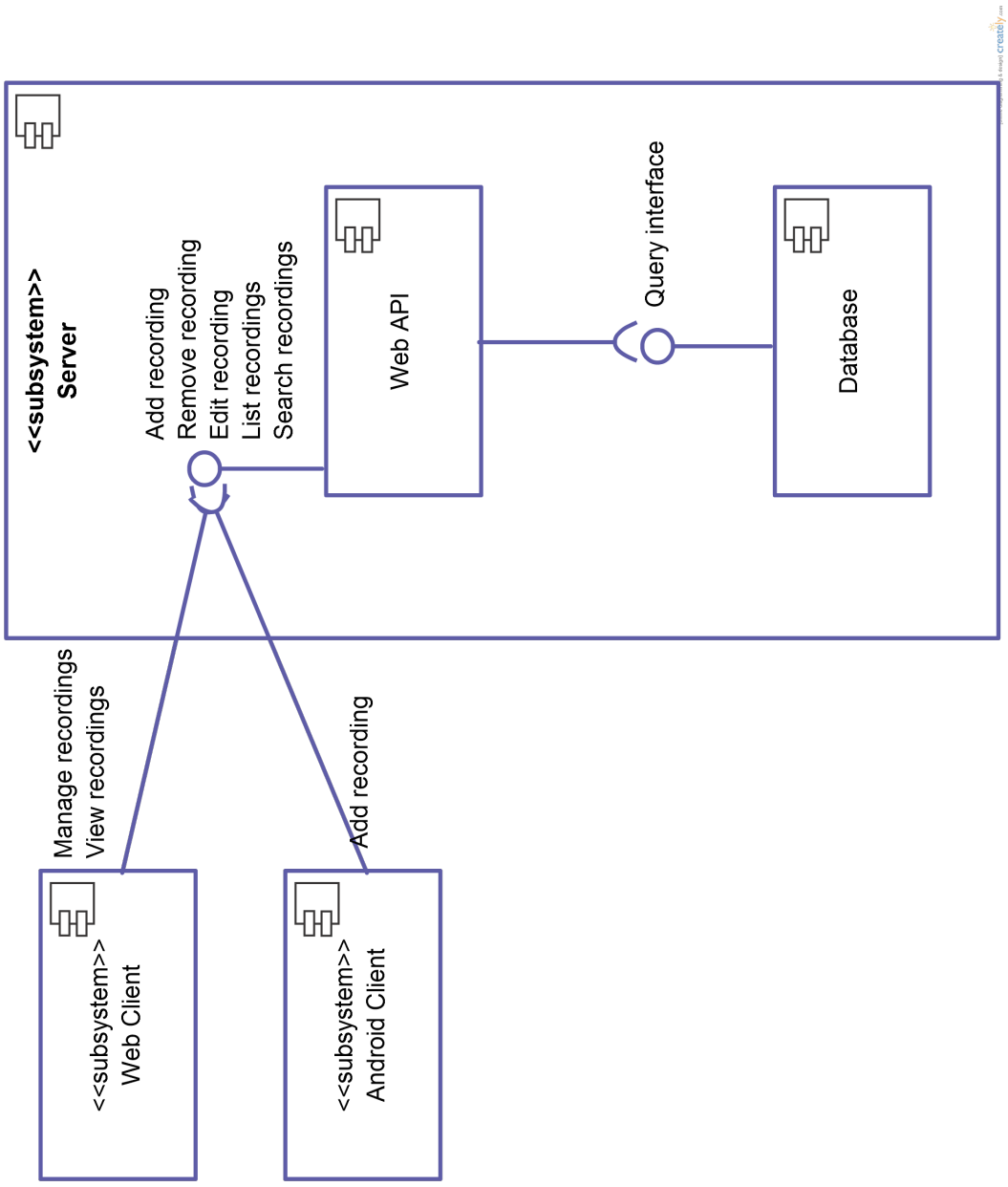



Figure 4: Web Api component Diagram

### 4.2.2 Significant data structures

The web API is going to make use of the Record and Specimen 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
  - Specimens : Array of Specimen

The structure of a Specimen is as follows:

- Specimen : Object
  - SpeciesName : String
  - LocationLatitude : Number
  - LocationLongtitude : 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 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)

- Resources
  - ResourceId : INT auto-increment PK
- 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] Requirements Specification Document.

# 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 release | December 02 2014 | nid21 |