

# Reiman Gardens Butterfly Wing Web Application

## Final Report

### **Team Dec1608**

Michael Bonpua

Scott Mueller

Carson Noble

Megan Reiman

Nicholas Riesen

### **Adviser**

Dr. Diane Rover

### **Client: Reiman Gardens**

Nathan Brockman

Anita Westphal

6 December 2016

# Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Introduction</b>	<b>3</b>
<b>Requirements</b>	<b>4</b>
Functional Requirements	4
Visitor Requirements	4
Administrative Requirements	4
Non-Functional Requirements	4
Coding and Workflow Standards	5
Naming Conventions	5
Code Formatting	5
Documentation	6
Branching Policy	6
<b>Implementation Details</b>	<b>6</b>
Tools	6
Server Side	6
Client Side	7
Class Hierarchy and Design	8
Controllers	8
Models	9
Platform / Hosting	10
Operating environments	10
Testing processes and testing results	10
Code Review	10
User Acceptance Testing	11
<b>Appendix I: Operation Manual</b>	<b>12</b>
Home page	12
Stats Page	13
Butterfly Search	13
Gallery	14
Plants	15
Learn More	15
<b>Appendix II: Alternative Versions</b>	<b>17</b>
<b>Appendix III: Code</b>	<b>19</b>

Routes	19
Controllers	20
Blade Templates	20
Models	21

# Introduction

Reiman Gardens hosts a butterfly wing in which they house several hundred butterflies at a time for visitors to see. In an effort to effectively convey information about the various butterfly species, Reiman Gardens developed a web application to complement the experience.

The original application provided the ability for visitors to the wing to see images of the butterflies, facts about specific butterflies, and to search for butterflies based on traits. Visitors could use their phones or the in-wing kiosk to view the application's website. However, this initial application did not fulfill its original intentions and several desired features were left out of the completed application. There were also issues scaling the site to fit different sized screens, a large portion of the data was hardcoded and could not be easily edited, and the site overall was hard to navigate and full of bugs.

The goal of this project was to build a completely new application to replace the old one. The new application would provide all of the functionality the original was intended to have, as well as additional functions requested by the Reiman Gardens staff. The final product would allow visitors to view images, information, and statistics about butterflies, as well as search for them by traits or by name. It would also give the Reiman Gardens staff control over functionalities such as featuring specific butterflies and calculating flight statistics in the wing, plus back-end editing of butterfly data. Finally, they could use the site to track shipments of butterflies received into the wing.

# Requirements

## Functional Requirements

### Visitor Requirements

The site shall allow visitors to:

1. Look up a butterfly based on physical traits.
2. Look up a butterfly based on its scientific or common name.
3. Look up a butterfly based on whether it is currently present in the butterfly wing.
4. View real-time search results based on their search criteria.
5. View information about an individual species of butterfly.
6. View statistics about the current state of the butterfly wing.
7. View statistics in a graphical, aesthetically interesting manner.
8. Browse a photo gallery of butterfly images.
9. Browse images of butterflies currently present in the wing.
10. Browse educational articles.

### Administrative Requirements

The site shall allow administrative users to:

1. Log information about shipments of butterflies.
2. Import and export Excel files containing details about butterfly shipments.
3. Log information about butterfly releases.
4. Add, edit, and remove butterfly information.
5. Add and remove images of butterflies.
6. Set a featured “butterfly of the day”.
7. Enter articles to be displayed to visitors.
8. Enter data and have statistics generated based on this data to be displayed on the site for visitors.

### Non-Functional Requirements

The site shall:

1. Require administrative users to authenticate in order to access administrative functionality.
2. Hide a login option from visitors.
3. Be aesthetically pleasing to visitors to the butterfly wing.

4. Render well on all screen sizes.
5. Be intuitive to use by visitors and volunteers of all ages.
6. Contain a page for adding butterfly release information that is simple and easy to use, especially from a mobile device.

## Coding and Workflow Standards

### Naming Conventions

All PHP and Javascript files must adhere to the following code-naming conventions.

Naming Style	Capitalized	Pascal Case	Camel Case
Example	EXAMPLE_NAME	ExampleName	exampleName
Applies to	Constants	Classes	Method / Function names Class attributes Method / Function Parameters

*Table 1: Naming standards*

### Boolean Test Methods

Methods that perform a simple test and return a Boolean value should start with the word “is”. For example, a method that indicates whether a container is empty or not should be “isEmpty()” rather than simply “empty()”.

### Getters and Setters

Attributes may be protected only if there is a compelling reason such as being required by the laravel framework. All changes in value to an attribute must be through some kind of method or changed by a subclass. Methods that either set or get the parameter of an object with minimal processing must have the function name start with “set” or “get” respectively.

### Code Formatting

The default conventions are derived from the PhpStorm auto formatter utility. Developers should run this utility on any file they modified prior to submitting their code for review.

## Documentation

All classes, methods, and functions created must have a phpdoc. The document must clearly state what the class, method or function is for as well as what the parameters are and the return value if they are present. PHP docs must be full sentences and grammatically correct. It is ultimately up to the person performing the code review to determine if these conditions have been satisfied.

If documentation is absent from a file that was modified, for whatever reason, the developer to last work on that method should add that documentation. The documentation will be added on the branch in which the documentation was discovered missing and submitted back to the original reviewer for approval.

## Branching Policy

No code will ever be committed directly to master. For each task, a new branch will be created and named according to the goal of the task to complete on that branch. The developer will do only the work for that specific task on that branch.

When development is done, the developer should merge from master into the working branch and resolve any merge conflicts. The reviewer, once satisfied, will merge the working branch into master.

# Implementation Details

## Tools

### Server Side

For the backend, we used the PHP framework Laravel. This framework provided us with a number of useful utilities and libraries that made development of the project and structuring the code faster and cleaner. Some of the utilities that were particularly useful to us were the web page templating, object-to-database mapping, authentication services, and validation services.

Laravel allowed us to closely follow the MVC pattern and to clearly define in what layer a specific functionality belonged. Maintaining this pattern meant that the codebase is well organized and easier to navigate and update.

Additionally, we used Laravel Excel, a utility by Maatwebsite. It enabled us to provide functionality for importing and exporting Excel documents of shipment data.

## Client Side

The biggest non-functional requirement for this project was for web pages to be dynamically sized to fit the screen on which they were displayed. To solve this problem, we employed Twitter Bootstrap, a CSS and JavaScript library that performs much of the screen sizing automatically. Bootstrap is based off of a 12-column grid system. As the screen size gets smaller, Bootstrap drops some of the extra columns down to a new row. In this way, we only needed to lay out our elements using the grid to ensure responsiveness.

Bootstrap also provided us with many common UI elements such as tabs and menu items.

The project also makes use of jQuery for manipulating page elements on the client side and making Ajax calls to the server. Examples of its use within the application are the search page and on the shipments pages.

Charts.js was used for generating both line chart and the pie chart on the stats page of the application.

On the gallery pages, LightBox was used to display a full sized image when an image on the page is selected, without having to load a new page for the image.

On the administrative side, TextAngular was used to provide a WYSIWYG text editor for use by the client to enter notes for the wing.



# Class Hierarchy and Design

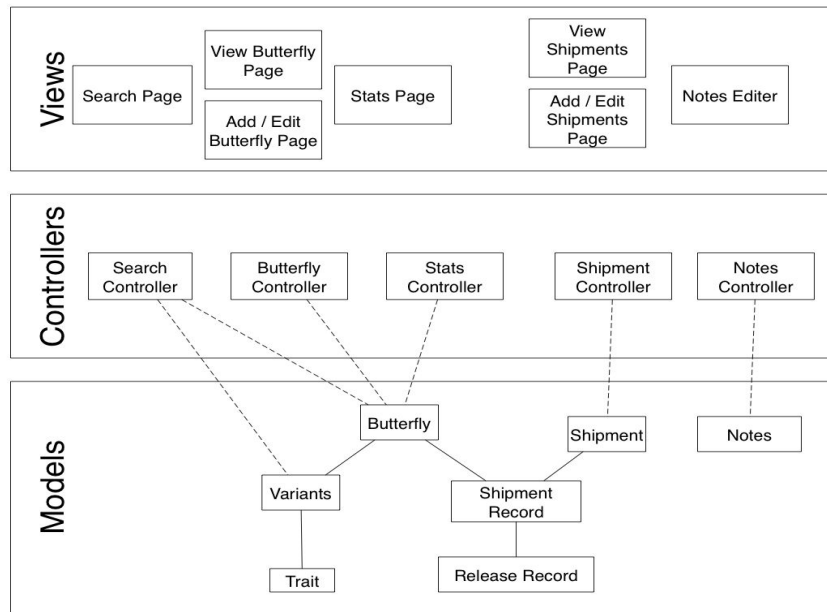


Figure 1: Class hierarchy displayed in MVC layers

As mentioned in the tools section, our project follows the MVC pattern. Laravel provides base controllers and model classes for the project classes to extend. The HTML views are generated dynamically by the Laravel blade engine.

## Controllers

**Search Controller** - The search controller was responsible for generating the SQL queries to perform searches requested by the user, and processing the data into a JSON form before sending it back to the user. Because traits are associated with a variant rather than the butterfly itself, the search operation actually searches for matching variants and returns their associated butterfly species.

**Butterfly Controller** - This controller is responsible for manipulating the Butterfly objects. Add butterfly, edit butterfly, delete butterfly operations all pass through this controller. This controller will also return the page to view information on a specific butterfly to the user.

**Stats Controller** - The stats controller is responsible for querying the database for and aggregating statistics about the butterfly wing as a whole. (Individual butterfly statistics are handled by the butterfly model.)

**Shipment Controller** - This controller processes all Shipments, ShipmentRecords, and ReleaseRecords. When a release or shipment is recorded, the new release or shipment will be handled here.

**Notes Controller** - This controller is responsible for processing notes Nathan likes to display on the site that may highlight a butterfly or an interesting fact in the wing. Created notes are stored in the database by this controller. Stored notes are also updated, deleted, and set as active to be displayed on the Stats page through this controller.

## Models

**Butterfly** - This model represents a butterfly species and contains information about the species such as scientific and common name, longevity, food source, and range. The model is linked to different Variants of the butterfly and to ShipmentRecords which detail how many of the species have been received by the wing.

**Variant** - A model whose main role is to link the images and traits to a butterfly species. Some butterflies demonstrate dimorphism (when males and females look significantly different despite being the same species). Other times, a butterfly's traits may vary based on the region. This intermediate model allows for different sets of images and traits to be mapped to one butterfly species.

**Image** - Represents an image of a butterfly. It is mapped exclusively to a Variant object. The Images model contains information such as the image name and the type of image (wings open, wings closed, pupa, larva, etc.).

**Trait** - This model represents a Trait that a butterfly may display. Such traits include inside and outside wing colors, the number of walking legs, the butterfly size, and food source. The search query attempts to find butterfly variants that map to as many of the requested traits as possible.

**Shipment** - Represents a shipment from one of Reiman Gardens suppliers. This model includes the date the shipment was sent, the date it arrived, and the supplier it came from. It then links to several ShipmentRecords which contain the information about individual butterfly species.

**ShipmentRecord** - Represents the part of a shipment related to an individual butterfly species. The model includes the number of butterflies received and information about how many were damaged in transit, etc. It is linked to the overall Shipment and to the Butterfly.

**ReleaseRecord** - Represents the subset of butterflies from a ShipmentRecord which were released into the wing on a given day. The model includes the date of the release and the number of butterflies released. It is linked to a ShipmentRecord.

**Note** - Represents a note that may be posted by an administrator. The note contains text to display to the user as well as an indicator signifying if the note should be displayed. The active notes are displayed on the homepage.

**Plant** - Represents a plant species. This will eventually host a similar structure to the Butterflies relations with its associated classes. This is an area for future development.

## Platform / Hosting

The application runs on an Apache HTTP server configured to run PHP with the Laravel framework. The server makes use of a MySQL Database. Both the server and database are run on an Ubuntu server. The server is provided by the cloud hosting service DigitalOcean.

The code repository is hosted in a private GitHub under the Reiman Gardens account.

## Operating environments

The application is run on a web server and used from the following platforms:

- Kiosk in the butterfly wing
- Volunteers'/staff members' tablets
  - Within the wing
  - In the lab
- Visitors' phones

## Testing processes and testing results

### Code Review

#### **The Developer's Responsibilities**

The developer is ultimately responsible for every aspect of their code. The reviewer is simply a check against mistakes and to check that the code is understandable by someone other than the original developer. As such, the developer must test their code and verify that the code works as intended. Developers should also check that the code is easy to understand and well commented. No commented out code is permitted.

### **The Reviewer's Responsibilities**

The reviewer's primary responsibility will be to verify that the code is maintainable. This includes verifying the code adheres to the agreed upon naming and formatting conventions, and that the code is easy to read and follow logically.

The reviewer should verify that all aspects the task were addressed by the submitted code, and ask the developer about any points of concern in the code.

It is not the responsibility of the reviewer to extensively test the code, although they may if they have concerns about portions of it. It is the responsibility of the developer to thoroughly test the code prior to submitting it for review.

### **User Acceptance Testing**

A test server is maintained to verify functionality in an environment identical to that of production. The environment is available to both the client and the developers.

# Appendix I: Operation Manual

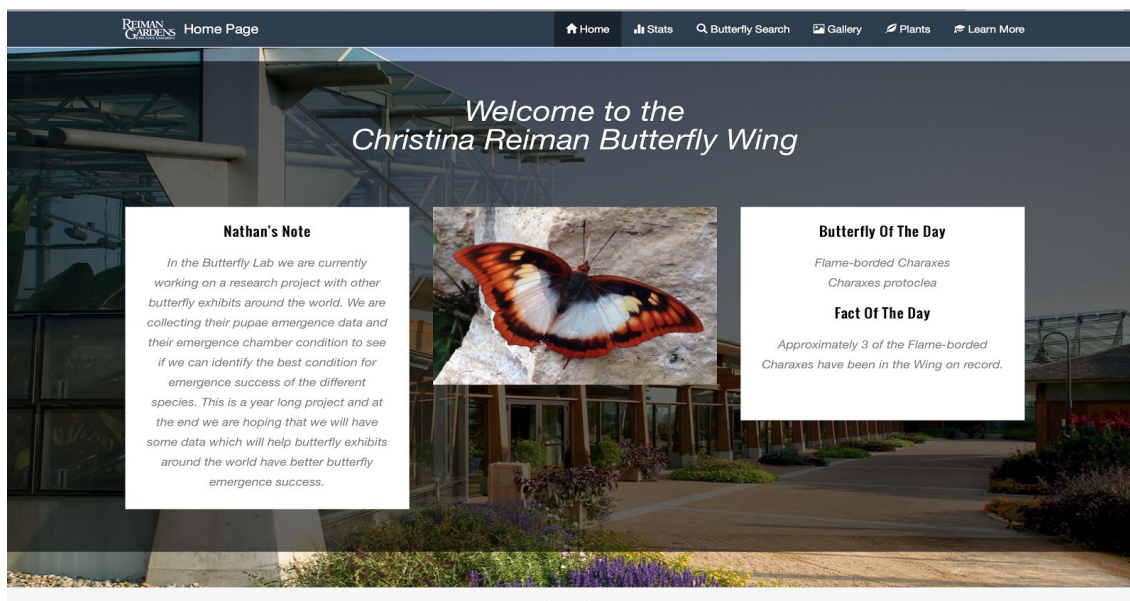
## Home page

Open a web browser (Chrome, Safari, Firefox, etc.)

Copy and paste the following into the address bar: <http://162.243.245.97/>

(note: this is the IP address of the test server)

That will bring you to the home page of the site, which will look similar to the following:



*Figure 2: A screenshot of the home page as it appears to visitors on the kiosk*

From the home page, you can browse to any of the other pages of the site using the navigational tabs at the top right. The butterfly of the day and Nathan's Note are displayed on this page as well.

# Stats Page

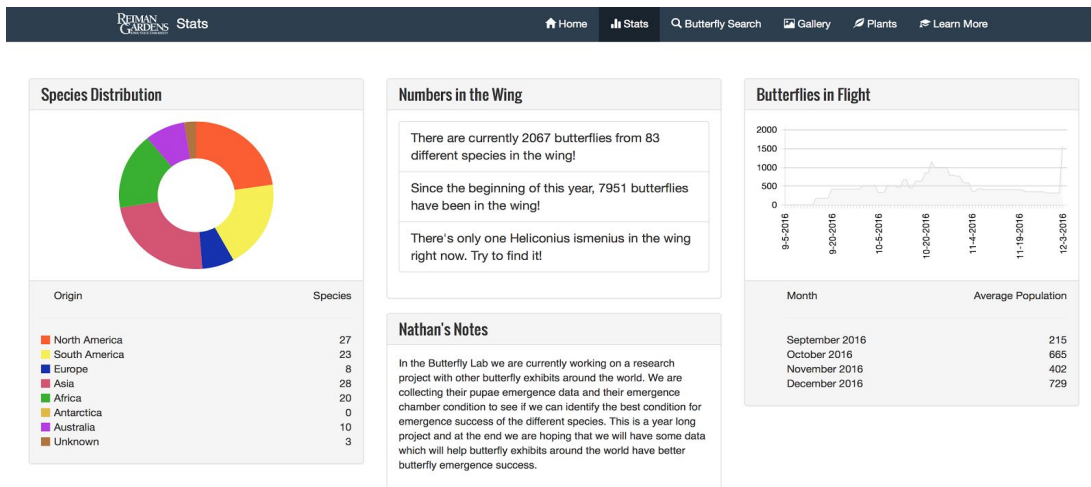


Figure 3: A screenshot of the Stats Page

The statistics page shows fun facts and graphs about the current state of the wing. Such information as the historical butterfly population in the wing, and the region distribution from which the butterflies can be seen is shown here.

The page has limited interactivity, but you can hover over the species distribution chart to see a label for each section.

# Butterfly Search

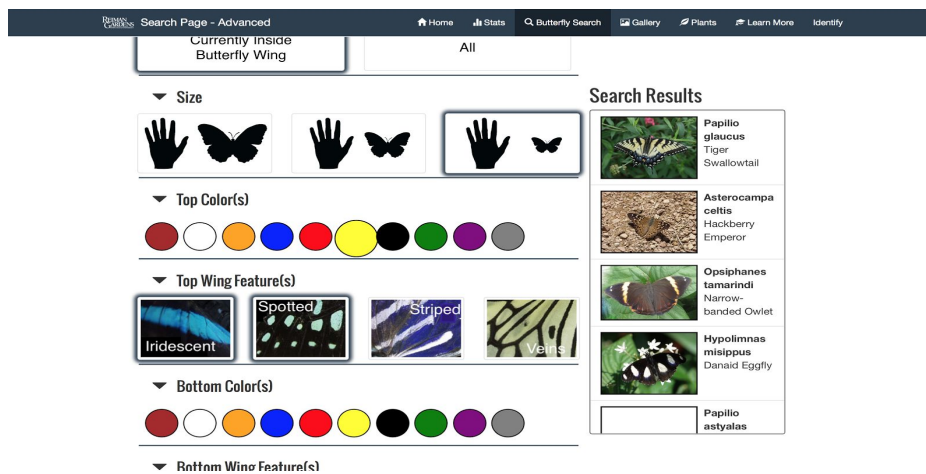


Figure 4: A screenshot of the Advanced Search Page

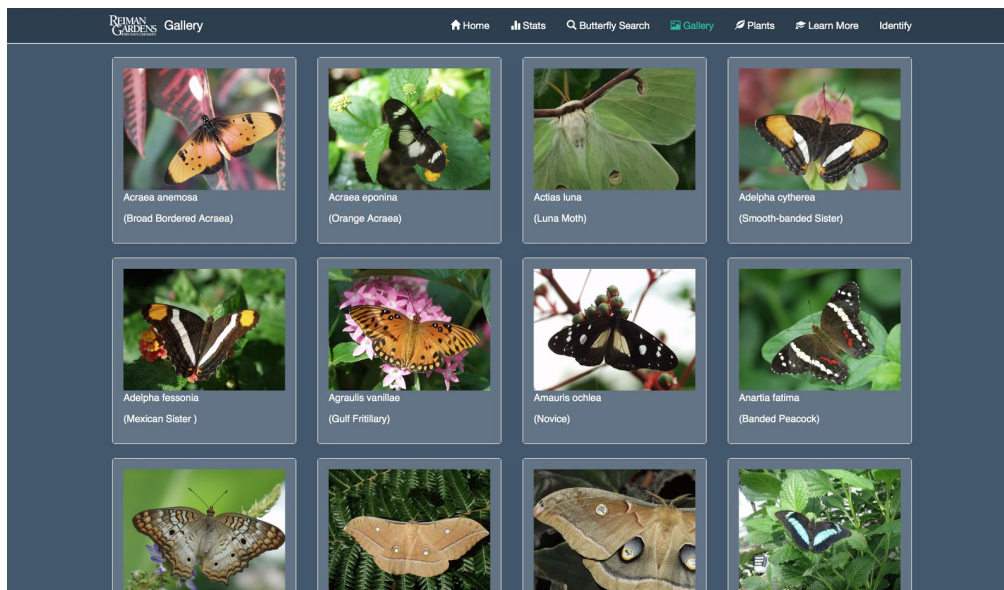
The search page allows guests of Reiman Gardens to look up a butterfly based on traits and name. The search field is populated dynamically as a user selects traits.

You can tab between the basic and advanced search pages for more search options.

To select a trait, click on the image that matches what you want to search for.

For more information about a butterfly, click on it in the Results panel.

## Gallery



*Figure 5: A screenshot of the Photo Gallery*

The gallery page allows a user to browse and view different pictures of the butterflies.

You can click on an image to expand it, or click on the name of a butterfly to see more information about that species.



# Plants

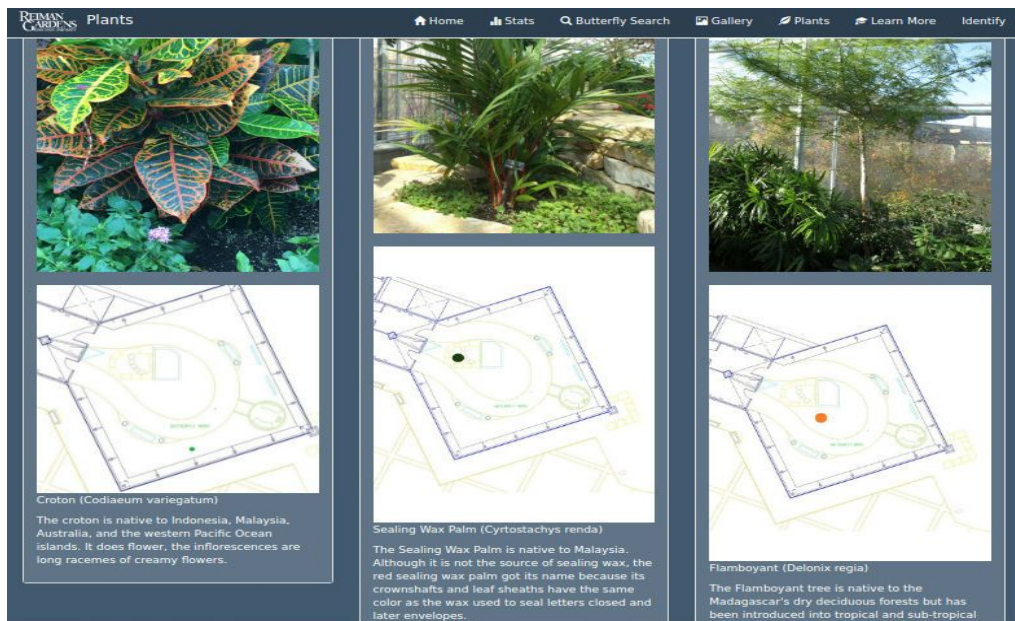


Figure 6: A screenshot of the Plants Page

# Learn More

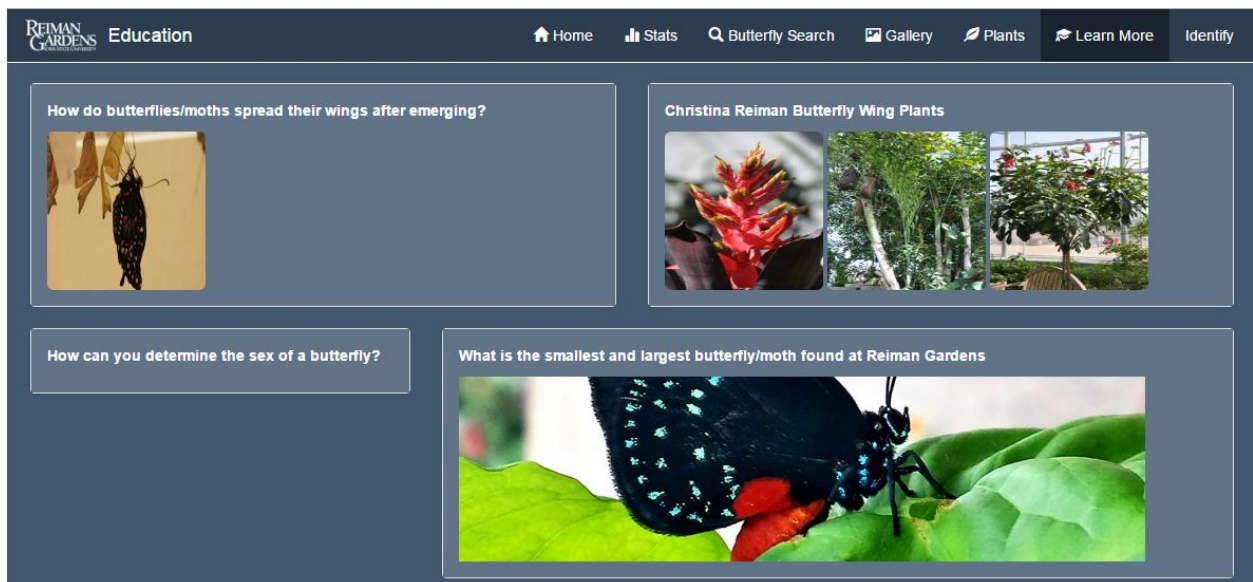


Figure 7: A screenshot of the articles browser



The learn more page contains links to articles where visitors to Reiman Gardens can go to learn more about a particular topic.

You can click on one of the article tiles to see that article. An example article is shown below.



Figure 8: A screenshot of an article

## Appendix II: Alternative Versions

Initially several different design layouts based on the previous project were created. Upon meeting with the client and hearing their concerns, some of these ideas were scrapped in favor of what has been developed to date. In general, these versions were thrown away as they did not meet the client's desire to focus the gallery and daily statistics about butterflies in the wing.

These ideas were also impractical after we learned more about the project. The first version was very focused on the front end to show off to the client what the new website could look like. As our knowledge of what was desired grew, we learned that these designs provided essentially no functionality as they were static HTML pages. We then moved the front end work to the backlog and focused on designing a backend and database that would allow us to provide functionality to the client.

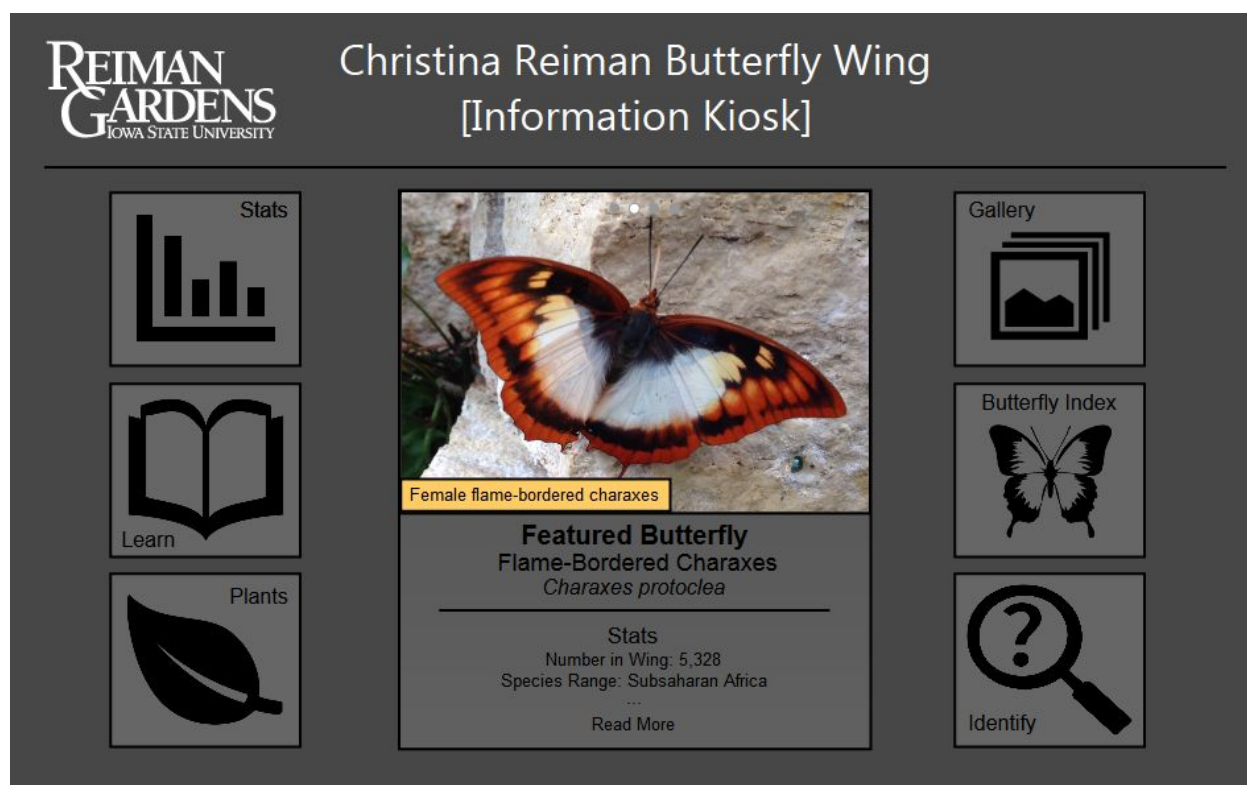


Figure 9: The initial idea for the Home Page


# Welcome to the Reiman Gardens Butterfly App!

### Butterfly Garden Stats

Total butterflies in wing:  
Species in wing:  
Most common country of origin:

[See more stats](#)

### Featured Butterfly



[New Guinea Birdwing](#) [Butterfly Gallery](#)

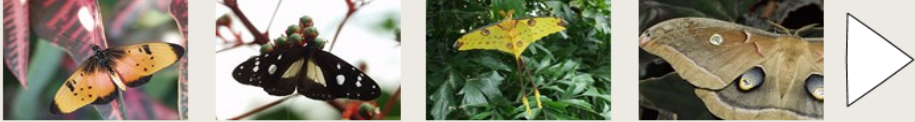
### Search Butterflies

Colors

Wing Size  [Advanced Search](#)

### More Places

- Garden Hours
- Plant Database
- Scan QR Code



[Go to Gallery](#)

Figure 10: An alternative layout idea for the Home Page

# Appendix III: Code

## Routes

Laravel gave us the ability to define routes within our site and direct them to particular controller methods. We simply specified the type of request we expected to a given route, then the name of the controller and the method to call. That also made it easy to include middleware (ex. for authentication) for individual routes.

```
21 Route::get('/stats', 'StatController@showAll');
22
23 // Search Routes
24 Route::get('/search-basic', 'SearchController@getBasicSearchPage');
25 Route::get('/search-advanced', 'SearchController@getAdvancedSearchPage');
26 Route::get('/search/results', "SearchController@searchResults");
27 Route::get('/butterfly/{scientificName}', 'SearchController@getButterflyPage');
28
29 Route::post('/butterfly/edit/{id}', 'ButterflyController@getID');
30
31 // Articles Routes
32 Route::get('/articles', 'ArticlesController@showAll');
33 Route::get('/articles/{id}', 'ArticlesController@show');
34
35 // Gallery Routes
36 Route::get('/gallery', 'GalleryController@firstPage');
37
38 //Administrative Views
39 Route::get('/shipments/butterflies', 'ShipmentsController@getButterflies')->middleware('ajax');
40 Route::get('/add_shipment', 'AdministrationController@addShipmentPage')->middleware('auth');
41 Route::get('/shipments', 'AdministrationController@shipmentsPage')->middleware('auth');
42 Route::get('/shipments/{id}', 'ShipmentsController@show')->middleware('auth');
43 Route::get('/shipment/edit_good/{id}', 'ShipmentsController@editGood')->middleware('auth');
44 Route::get('/shipment/edit_poor/{id}', 'ShipmentsController@editPoor')->middleware('auth');
45 Route::get('/shipment/edit_none/{id}', 'ShipmentsController@editNone')->middleware('auth');
46
47 Route::post('/shipments/new', 'ShipmentsController@addShipment')->middleware('auth');
48 Route::post('/shipment/update_good/{id}', 'ShipmentsController@updateGood')->middleware('auth');
49 Route::post('/shipment/update_poor/{id}', 'ShipmentsController@updatePoor')->middleware('auth');
50 Route::post('/shipment/update_none/{id}', 'ShipmentsController@updateNone')->middleware('auth');
51
52 //Notes Editor
53 Route::get('/notes_editor', 'NotesController@notesEditorPage')->middleware('auth');
54 Route::put('/notes_editor', 'NotesController@update')->middleware('auth');
55 Route::post('/notes_editor', 'NotesController@store')->middleware('auth');
56 Route::delete('/notes_editor', 'NotesController@remove')->middleware('auth');
57
58 //Butterfly Entry Routes
59 Route::get('/add_butterfly', 'AdministrationController@addButterflyEntryPage')->middleware('auth');
60 Route::get('/edit_butterfly', 'AdministrationController@selectButterflyEntryPage')->middleware('auth');
61
62
63 Route::post('/add_butterfly', 'ButterflyController@addButterfly')->middleware('auth');
64 Route::post('/edit_butterfly', 'ButterflyController@getButterflyID')->middleware('auth');
65 Route::post('/edit_butterfly/{id}', 'ButterflyController@editButterflyPage')->middleware('auth');
66
```

Figure 11: The route.php file linking html routes with controllers

## Controllers

Controllers are used to group related route logic into one class. For instance, the butterfly controller handles all of the back end functionalities for adding, editing, and deleting butterflies. The butterfly controller handles a total of seven routes for all butterfly edits made by administrators.

```
class ButterflyController extends Controller
{
    /** Gets ID from search page and returns to edit butterfly page
     *
     * @param Request $request from selectButterflyEntry.blade.php
     * @return butterfly edit page
     */
    public function getID(Request $request) {
        $currentbutterfly = Butterfly::find($request->id);
        return view('pages.administration.editButterflyEntry', compact('currentbutterfly'));
    }

    /** Gets ID from select page and returns to edit butterfly page
     *
     * @param Request $request from selectButterflyEntry.blade.php
     * @return butterfly edit page
     */
    public function getButterflyID(Request $request) {
        $currentbutterfly = Butterfly::find($request->editButterflyEntry);
        return view('pages.administration.editButterflyEntry', compact('currentbutterfly'));
    }
}
```

Figure 12: The ButterflyController class

## Blade Templates

Blade is Laravel's templating engine, which is used to maximize code reuse. It enabled us to create an overall template for the layout of the application (app.blade.php) and extend it for each individual page in the site.

Blade also allows us to reuse common UI elements. Some of these elements include the collapsible headers used on the search page, and the image buttons used on the search page and the add butterfly page.



```

1
2 @extends('pages.searchPage.searchPage-sharedLayout', ['pageTitle' => 'Search Page - Basic'])
3
4 @section('searchOptionsPanel')
5 <ul class="nav nav-tabs row">
6 <li class="active"><a href="#">Basic Search</a></li>
7 <li><a href="{{ action("SearchController@getAdvancedSearchPage") }}">Advanced Search</a></li>
8 </ul>
9
10 @include('layouts.UI-Elements.searchHeading', [
11 'name' => 'name',
12 'heading' => 'Butterfly Name'
13 ])
14
15 <div id="nameOptions" class="input-group">
16 <span class="input-group-addon" id="basic-addon1"><span class="glyphicon glyphicon-search"
17 aria-hidden="true"></span></span>
18 <input id="searchByName" onkeyup="dynamicSearchAdvanced()" type="text" class="form-control"
19 placeholder="Butterfly Name" aria-describedby="basic-addon1">
20 </div>
21
22 @include('layouts.UI-Elements.imageButtonGroup', [
23 'name' => 'inWing',
24 'heading' => 'Inside Wing',
25 'function' => 'radioButton',
26 'default' => 'true',
27 'onSelect' => 'dynamicSearchBasic',
28 'options' => [
29 ['image' => 'img/search/presenceState/inside.png', 'value' => 'true'],
30 ['image' => 'img/search/presenceState/all.png', 'value' => 'false']
31 ]
32 ])
33
34 @include('layouts.UI-Elements.imageButtonGroup', [
35 'name' => 'size',
36 'heading' => 'Size',
37 'function' => 'radioButton',
38 'onSelect' => 'dynamicSearchBasic',
39 'options' => [
40 ['image' => 'img/search/sizes/Large.png', 'value' => 'large'],
41 ['image' => 'img/search/sizes/Medium.png', 'value' => 'medium'],
42 ['image' => 'img/search/sizes/Small.png', 'value' => 'small']
43 ]
44 ])

```

Figure 13: The Basic Search Page blade file

## Models

Models in Laravel provide an interface to the database using Eloquent, Laravel's ORM (object-relational mapping). Model classes are created in the app directory of the project, and each one is associated with a table in the database. They provide methods for viewing and changing elements within the related table. For instance, our database includes a table for shipments. The table is linked to shipment\_records (another table related to the butterflies of a particular shipment). The Shipment and ShipmentRecord models allow us to get the shipment records associated with a given shipment (and vice versa) through a simple method call (see the method shipmentRecords() in the code below). They also give us the ability to create shipment records to be associated to the given shipment without having to deal with matching IDs between the two database tables.

```

class Shipment extends Model
{
    public $timestamps = false;
    public $incrementing = true;

    // Fields that can be edited by the user
    protected $fillable = [
        'shipment_date',
        'arrival_date',
        'supplier'
    ];

    /** Defines relationship between Shipment and Shipment_record
     *
     * @return \Illuminate\Database\Eloquent\Relations\HasMany
     */
    public function shipmentRecords() {
        return $this->hasMany(ShipmentRecord::class);
    }
}

```

*Figure 14: The Shipment model*