Global Butterfly Longevity Tracker

Design Document

sdmay25-03

Client- Nathan Brockman

Advisor- Maruf Ahamed

Team Members/Roles- Alex Herting(Frontend Developer), Andrew Ahrenkiel(Full Stack Developer), Carter Awbrey(Backend Lead), Charles Dougherty(Frontend Developer), Jaret Van Zee (Backend Developer)

Team Email- sdmay25-03@iastate.edu

Team Website- https://sdmay25-03.sd.ece.iastate.edu/

Date: 05-04-2025

Version 1.2

Executive Summary

The Reiman Gardens and other facilities with butterfly enclosures need a way to quickly and reliably store and access butterfly tagging data electronically. It is important that this data is readily available to facility managers because it can help them optimize their enclosures to maximize the butterflies' life spans. We are creating a web application that allows guests and facility docents to be able to enter butterfly sightings while they are at an enclosure. The facility admins will then be able to generate meaningful reports(life span, number of sightings, time since last sighting, etc.) based on all of the data collected from the sightings. Our application will be able to adapt to various butterfly exhibits and will allow each exhibit to use their own unique tagging system in order to fit their needs.

For our frontend, we are using HTML/CSS/JS and will not be using a framework in order to maximize performance. For the backend, we will be using the Spring framework; for our database, we will be using MongoDB; and for our server, we will be hosting on AWS. We are currently on the verge of deploying our application onto our client's AWS to begin beta testing with real guest users within the Reiman Gardens environment. This will allow us to get real feedback from not only the client, but guests and docents alike to maximize the user experience of the application moving forward. Our web application is readily available to use on devices of all sizes, from phones to desktop computers. We have ensured that our pages can adapt and still be presentable under all screen sizes and in multiple browsers. All admin functionality has been completed, which includes functionality for any tag definition a facility may be using in correlation to all web page views. Our next steps are to finish the functionality for administrator reporting, which includes a variety of unique data reports and multi-facility data comparisons. From there, we will continue our beta testing and increment to achieve the best user experience.

Learning Summary

1. Development Standards & Practices Used

a. **ISO/IEC 27001:** Information Security Management Systems

b. **ISO/IEC 25059:** Systems and Software Quality Requirements and Evaluation

c. **ISO/IEC 19772:** Authenticated encryption

d. **RFC 7231:** Hypertext Transfer Protocol (HTTP)

e. **RFC 8446:** Transport Layer Security (TLS) Protocol

f. **ECMA 404:** JSON Data Interchange SyntaxRFC 7510 - JSON Web Signature

g. **NIST SP 800-63B:** Digital Identity Standards

2. Summary of Requirements

The website application is required to meet several outlined requirements that were set by the client. Additional requirements were derived from the original functional and resource requirements.

The functional requirements included having a user login system (authentication), supporting a user hierarchy system, creating reports from queried data, quick response and query times, and for portability of the website. The client also requested that this web application be hosted on Amazon Web Services with a low upkeep cost. For storing butterfly data across several different butterfly facilities, several different data rules and requirements were created. These rules include enforcing data integrity policies, quickly accessing data, and ensuring data scalability. Several user experiences were laid out since this website will also be accessible to visitors of the butterfly facility. These requirements would ensure that the website is aesthetically pleasing to view, accessible on cellular devices, and easy to navigate.

3. Applicable Courses from Iowa State University

This is a list of applicable courses from Iowa State University and their direct application: SE 309–Concepts of software development as a team, SE 319–web development concepts applied to the web application, COM S 363–Database management and storage concepts applied to the database management system, SE 185–basic programming concepts applied to our code practices, SE 186X–Basic concepts of developing software as a team, CPR E 230–computer networking principles applied to the development of the web server, CPR E 231–cyber security principles applied to our overall system, SE 317–software testing concepts that were integrated into our development process, SE 421–security practices applied to our system, COM S 252–Linux operating system concepts applied to our virtual machines, COM S 352–operating system concepts applied to our deployment virtual machines, ENGL 314–technical writing concepts applied to our professional communication and documents, SP CM 212–public speaking skills applied to our relationships with our client, COM S 227–general programming and object concepts applied to our coding practices, COM S 228–general data structure concepts applied to our coding and efficiency practices, COM S 422–cloud computing concepts applied to our deployment system inside the cloud, COM S 417–specialized testing procedures applied to testing the web application against the requirements.

4. New Skills/Knowledge acquired

This project required our team members to take time to learn several new skills for the overall website to be built. Our client required the application to be hosted on AWS. However, our team did not possess the necessary experience to properly use the AWS features. To compensate for this, some team members took the time in order to learn how to integrate different resources from AWS into our application and how to properly host the web application on AWS.

At the beginning of the project, our client had sent us a Figma board that contained different web pages that were designed to his liking. He requested that we design our website to look like the web pages outlined in the Figma board. In a team decision, we decided that it would be best to learn how to import the existing designs on the Figma board into actual HTML/CSS files. The frontend team members took the time to learn PxCode, a software that allows you to transfer Figma board pages into HTML/CSS files while retaining the original design and features of the Figma board web page.

Spring was selected as the framework to build the backend on. This was because our team had already gained experience using this framework from previous projects. However, the complexity of the project required additional knowledge of using the Spring framework and integrating APIs into the system. This knowledge allowed the backend to be more aligned with the requirements.

In the application, a user account and login management system were required. Since this will be deployed out to the public through butterfly facilities. Putting an emphasis on security for the accounts is required. Using JSON Web Tokens (JWT) was decided as the best approach to implement authentication in the web application. The backend members had to learn how to set up and integrate the authentication into the existing API calls.

The selected method of data storage was by using MongoDB, a non-relational database. However, using the built-in MongoRepository library inside Spring with the basic query calls would cause requests to take longer than anticipated. Several requests would cause response timeouts. In order to expedite the database queries, members on the backend had to integrate several advanced Mongo queries and aggregation pipelines. By implementing the advanced queries and pipelines, this allowed for the database query request times to be within the time specifications that were set by the client.

Table of Contents

1. Introduction

1.1. Problem Statement

The Reiman Gardens and other facilities with butterfly enclosures need to be able to quickly and reliably store and access butterfly tagging data electronically. Butterflies within enclosures are currently being tagged; each facility needed a standardized way to enter and store butterfly sightings relative to their facility's tagging system. Facilities need a web application that allows guests and volunteers to quickly enter a butterfly tag sighting, as well as gives facility administrators a fast and reliable way to view database information. Reporting data is essential for facilities to make educated decisions on enclosure environments and derive conclusions based on certain butterfly species' lifespans.

We developed a web application to address the needs of any facility with a tagged butterfly enclosure. The web application provides an easily accessible user interface from any device in order to provide the highest level of sighting entries from any type of user. Our solution allows administrators to build reports on this sighting data dynamically, with customizable report types and filters.

1.2. Intended Users

There are four tiers of users for our application, each of which has different roles and privileges within the application. Each of those users is listed below.

Guests: A guest is a person who is visiting the butterfly enclosure for educational or entertainment purposes. A guest visitor could be a person of any age and background. Guests need an easy-to-use web application that they can use to quickly enter a butterfly sighting while visiting a butterfly enclosure. Since guests will be using a mobile device to enter butterfly sightings, they need a web application that can be used on any type of device.

Docent / Volunteers: A docent or volunteer is a facility member who regularly spends time within the butterfly enclosure. Since a docent will use the web app much more frequently, they need a fast and easy way to continuously enter sightings. Docents will also benefit from an individual login so their sightings can be entered with more credibility than a guest visitor. Since docents are more reputable and garner more priority, docent accounts may also need to be configured with higher levels of privilege to database information based on administrative needs.

Domain Administrator: Domain Administrators will be the person or people in charge of the butterfly enclosure at their given facility. A domain administrator will be a person with a higher level of knowledge of butterfly species, as well as their given enclosure and tagging system. Domain administrators will need to configure the web app for their facility and its tagging system, as well as configure user accounts for docent and guest users. Domain administrators will also need full access to the database of butterfly sightings and need to be able to generate reports based on their butterfly sightings.

Super Administrators: Super Administrators will be above all facilities and be able to create new facilities based on the growth of the application. Super Administrators will also be able to view and report on all data regardless of which facility it is from, in addition to all permissions of the other user groups.

2. Requirements, Constraints, and Standards

2.1. Functional and Non-Functional Requirements

1. User Authentication

The user website is required to have a user authentication method where administrator and docent accounts can securely log in or be created. A basic password recovery and reset functionality will be available only to the Super Admin. Accounts can be registered with a unique username and password. Guest users will not require a password when their account is created to tie butterfly sightings to.

2. Support User Group Hierarchy

The user account type and permission scheme will be divided into four different levels, the order from lowest to highest permission level: Guest User, Docent, Domain Admin, Super Admin. The Guest User needs to be able to submit the butterflies that they see during their visit within the web application, because it updates butterfly information. The Docent needs to be able to easily insert butterfly sightings while maintaining a high level of credibility through secure credentials. The Domain Admin needs to be able to view data from their site because they want to be able to use the data from their site for research. The Super Admin needs to be able to view all data from every facility in order to create reports that compare multiple facilities' data.

3. Support Multiple Domains/Facilities

The web application must be scalable to support the ability to add multiple butterfly domains/facilities through the approval of the Super Admin. Each domain will have its own set of butterfly tagging rules. Each domain's butterfly and user data will be isolated from other domains. However, it will be accessible by the Super Admin. Each Domain Admin can manage their own domain but cannot access other domains.

4. Create Reports from Queried Data

A Domain Admin or Super Admin has the ability to generate specialized butterfly reports based on the butterfly data in their facility. Some of these reports include calculating the average lifespan by butterfly species, finding the last sighting of an individual butterfly, finding the total number of sightings on a butterfly, calculating the total number of butterflies of a specific species, and any additional reports per the client's request.

5. Quick Response and Querying Times

The web application must be able to have very quick performance and response times. These requests that must have fast performance include generating butterfly reports and page load times. Some metrics outlined include having page load time below 2 seconds, interactive elements responding within 500 milliseconds from the click time, fetching data in under 4 seconds, and generating reports in under 4 seconds.

6. Portability

This website must be able to work on all basic platforms that have a web browser, i.e., desktop, tablet, and mobile devices. The website must be able to adapt its structure to fit the format of each of these devices.

2.2. Resource Requirements

1. AWS Hosting Services

The system will use AWS for hosting because the client is most familiar with AWS and wants to be able to consolidate all his services to a single provider.

2. Cost-Efficient Upkeep

The website should run for no more than $15 per month while maintaining a high level of performance, which is comparable to the previous designs.

2.3. Aesthetic Requirements

1. Color Scheme

The website will be designed with a consistent color scheme throughout the web pages. The color palette will be well-designed and follow basic color theory to enhance the user experience. The background and text will contrast with each other in order to ensure readability.

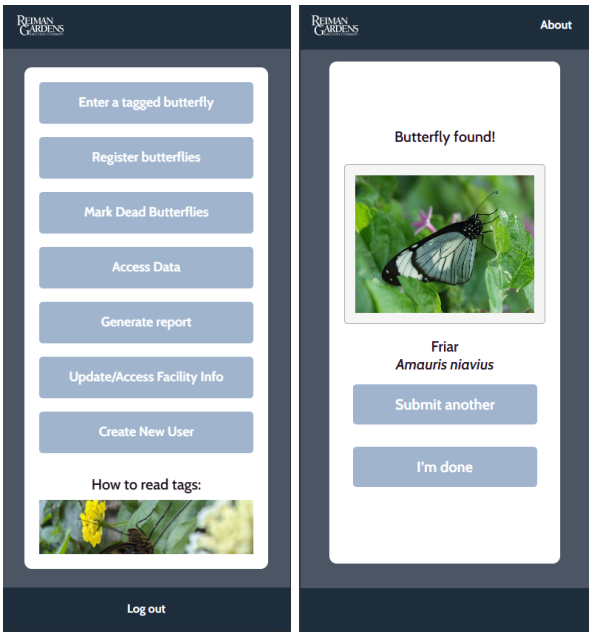


Figure 2.3 and 2.4 (Images from our website design that display color theme characteristics)

2. Typography

The website will use fonts that are compatible with all web browsers and that are readable for most users. The website will have adequate line spacing and margins for readability.

3. Images

All images that are used will be high-quality and high-resolution images that are related to the content.

2.4. User Experiential Requirements

1. Ease of Navigation

The website must be easily navigable and intuitive for the average user. This can be obtained by ensuring the website has a clear and logical structure. The website must have navigation elements to assist the user.

2. Accessibility

The website has several accessibility features for users when they access the website. The website will be easily accessible to any user with an internet connection or cellular data connection. This website will be compatible with guests on all types of mobile devices. If the internet or data connection is slow, alternate text will be provided for the images.

2.5. Database Requirements

1. Data Integrity

Data validation rules will be implemented to ensure that all data is accurate and reliable. Data must remain consistent throughout the use of constraints.

2. Scalability

The database must perform within the performance metrics, even as the amount of data grows progressively throughout the lifespan of the web application. The database should be able to scale and handle different domains being added to the system.

3. Performance

The required metrics for the database metrics have been outlined after agreement between the development team and the client. The database must

be able to complete queries in under 4 seconds. The database must complete GET, POST, PUT, and DELETE operations in under 4 seconds.

4. Design

The data must be designed in a way that will maximize the performance. The database design should allow for collections to be connected efficiently.

2.6. Engineering Standards

As we developed our project, we made a point to follow a few key engineering standards to help guide our design decisions, especially around how we structure our APIs, handle authentication, and protect user data. Below are a few of the most relevant standards we used, along with how they directly applied to different parts of our system. These standards helped ensure that what we were building wasn't just functional, but also secure, reliable, and in line with industry best practices.

1. RFC 7231 – HTTP/1.1 Semantics and Content

This standard defines how HTTP methods like GET, POST, PUT, and DELETE should behave, and how to handle things like status codes. Since our project is built around a RESTful API, this RFC helped us shape how our endpoints work. For example, we use GET to retrieve data, POST to create resources, and return appropriate status codes like 200 OK or 404 Not Found when things go wrong. Following this standard makes our API easier to understand and integrate with, both for developers and tools like Postman or Swagger.

2. RFC 7519 – JSON Web Tokens (JWT)

We use JWTs to manage authentication in our app — when users log in, they get a token that they can use to prove who they are on future requests. RFC 7519 defines how these tokens should be structured and signed. By following this spec, we made sure our tokens are secure and follow a widely accepted format. That way, even if our system grows or gets integrated with other services later, our authentication approach won't have to be completely reworked.

3. NIST SP 800-63B – Digital Identity Guidelines

This is a set of security recommendations from NIST (the U.S. National Institute of Standards and Technology) that focuses on how to safely handle things like passwords and user login flows. Based on this guidance, we made sure to hash all stored passwords using BCrypt, which adds both salt and computational difficulty to resist brute-force attacks. We also enforce decent password complexity and added rate-limiting to reduce the risk of automated login attempts.

3. Project Plan

3.1 Project Management/Tracking Procedures

We will be using Agile as our project management style for this project. The primary reasons that we decided on an agile approach were because of the flexibility, incremental delivery, continuous improvement, and collaboration aspects. Having independent tasks within our backlog is super important in software development. We do not want to have dependent tasks that could be delayed because of unexpected issues arising during development. We value the incremental delivery aspect of agile because we plan on producing prototypes for our client before providing him with the full version of the product. This ties into the continuous improvement aspect because as we produce prototypes, we will improve our product based on client feedback. The collaboration aspect of agile is excellent for software development. We have toned back the daily standup meetings to once a week to check in with one another and discuss the work we have done and plan to do.

We plan to track project progress using GitLab milestones. We have a GitLab repository set up with an issue board and milestones that are dated with deadlines. Inside GitLab, we have also set up a scheme for our branch setup to keep our code organized, allowing for easier project progression due to simplicity. Git is an amazing tool for software development because of its version control, and is the de facto industry standard for all software development. It allows us all to collaborate on our repository simultaneously and merge our progress together easily.

3.2 Task Decomposition

Frontend

The Frontend can be broken into two main tasks: the core HTML development and the backend interactions. The core HTML development involves creating the design and functionality of the web pages and focusing on the user experience. Completing the core HTML functionality can be broken down into several steps. The initial steps were to convert the client-provided Figma boards into functional HTML pages. Then, allowing each of those pages to be navigated properly. After the basic web page functionality is complete, additional functionality can be implemented. These additional functionalities would include the login screens, all associated database reporting screens, a facility management page with customization, modifying butterfly data screens, and registering butterflies. The last main HTML feature is to optimize the

web pages for desktop and mobile devices. Implementing backend interactions is required in order to access any data that is stored in the database. This will include authenticating users, sending requests for butterfly reports, and checking butterfly tag support.

Backend

The backend can be broken into several different services that need to be implemented in the web application. The first feature is a user management system. This allows for unique users to be created and stored in a database. Users can be later updated when necessary. The user management system integrates JWT authentication to enforce the permission scheme. A domain management system allows for multiple domains to be added to the overall system. Domains will have their own separate butterflies and users in the database. Users inside a domain can only access data that is related to their own domain, except for the Super Admin, who can access data from all domains. The Butterfly system will need a special universal tagging system. This tagging system will be compatible with almost all existing domains' tagging systems. Specialized queries will also have to be created for butterfly sightings. Report generation will also be implemented, which can be created based on parameters sent from the frontend. The backend will be hosted on a cloud infrastructure–AWS. This means that the cloud instance will have to be set up in a cost-effective manner that can still meet the requirements.

Frontend + Backend

The front and backend share similar tasks to each other. The main set of tasks that are shared is testing. This can include local development testing, manual UI testing, API testing, and any additional testing. Code reviews and merge requests are required on the front and back end as well. The final main shared task is to perform user testing with the currently available features in the app with actual test users in the butterfly domains.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Frontend

I. **UI Design Completion**

A. **Metric:** Percentage of originally requested designs successfully converted to functional HTML.

B. **Milestone:** Complete the conversion of 100% of requested views into functional HTML components.

II. **Responsiveness**

A. **Metric:** Whether a view is able to adapt to different screen sizes and aspect ratios while adequately displaying content.

B. **Milestone:** Ensure all implemented views adapt effectively to various screen sizes, including desktops, Phones, and Tablets.

III. **Compatibility**

A. **Metric:** Compatibility tests pass rate across target browsers.

B. **Milestone:** Ensure our designs achieve a 100% pass rate in compatibility tests across major browsers, including Chrome, Firefox, and Safari.

IV. **Functionality Development**

A. **Metric:** Percentage of interactive elements from the design that function as intended.

B. **Milestone:** Ensure all interactive elements are fully functional and have corresponding tests written.

V. **Accessibility**

A. **Metric:** Number of Web Content Accessibility Guidelines (WCAG) criteria met, as defined by W3C.

B. **Milestone:** Achieve an 'AA' level of accessibility by implementing the required WCAG guidelines.

VI. **Client Acceptance**

A. **Metric:** Level of client satisfaction with the design as measured through feedback.

B. **Milestone:** Achieve full client satisfaction with all designs, with no further changes requested after review.

Backend

VII. API Development and Integration

A. **Metric:** Percentage of API endpoints developed, tested, and documented.

B. **Milestone:** Implement 100% of API endpoints outlined in the project requirements, with thorough integration tests for each endpoint.

VIII. Database Performance and Optimization

A. **Metric:** Average database query response time for data visualization features.

B. **Milestone:** Achieve a response time that is at least 50% faster than the previous design while maintaining equivalent end-user functionality.

IX. Tagging Adaptability

A. **Metric:** Capability to integrate specific butterfly tagging systems into the database.

B. **Milestone:** Successfully incorporate all widely used butterfly tagging systems adopted by major institutions.

X. Security Compliance

A. **Metric:** Number of security vulnerabilities identified and remediated (tracked via penetration testing or security audits).

B. **Milestone:** Resolve 100% of critical vulnerabilities, aiming to avoid common security risks outlined by OWASP guidelines. Ensure that, to the best of our abilities, protections are in place against vulnerabilities such as broken access control, injection, cryptographic failures, and others.

XI. User Data Management and Compliance

A. **Metric:** Compliance with data protection standards as outlined in the GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act).

B. **Milestone:** Achieve compliance with data privacy and protection standards, ensuring that all user data is encrypted and anonymized where applicable.

XII. Error Handling and Uptime

A. **Metric:** Number of errors that impact user experience or server uptime.

B. **Milestone:** Implement logging and monitoring systems to ensure that the number of critical errors impacting user experience or server uptime remains at a minimum.

XIII. Data Safety and Recovery

A. **Metric:** Risk of data loss in the event of a system failure.

B. **Milestone:** Implement robust data backup procedures to ensure that no critical data is lost during a system failure. Validate backups and confirm the ability to recover data in the event of a failure.

3.4 Project Timeline/Schedule

3.4.1 Gantt Chart Tasks (Semester 1)

Figure 3.4.1 (Semester 1 Gantt Chart)

Converting the provided Figma board into HTML and CSS using a tool called PxCode. PxCode is a free tool that provides functionality for exporting Figma pages to HTML pages. Once pages have been exported to raw code, correct responsiveness, styling, page resizing, and page functionality must be implemented.

Creating the database management schema first involved brainstorming ideas for database collections and the overall layout. After 2 group meetings, we were able to design a database structure that worked for our data types and use case and began implementation.

When we were **creating the database objects**, we first created each collection with the object structures in mind. Then we defined each object and the parameters associated with the object in detail. This allowed us to have a baseline of what each object would look like and how to store them.

Containerization of application components is important for our hosting environment and the simplicity of use. We will create a container for both the front and back end components to be easily deployed onto our VM instance.

Mapping the backend to the database required us to structure the backend according to the database collections and object structures. We were able to define object parameters within the backend code to match the database structure. In order to confirm correctness, we created test requests to the database.

Multi-page navigation is not a feature of the PxCode Figma to HTML export. We will have to connect each HTML screen through JavaScript to ensure a proper user experience and application functionality. This also includes user interaction functionality with buttons, input fields, and more, which are not supported in the PxCode export.

Our **butterfly tagging system** needs to be adaptable to any facility using the application. This means each screen that allows for tag inputs must adapt to the facility's configured tagging scheme. UI components should be simple and straightforward for guests to use. Creating backed requests for sighting butterflies with a simplified methodology for any tagging definition is required for the overall functionality of the application.

The **database query system** was a complex task that took a lot of optimization and testing in order to meet the performance requirements. We first created baseline queries for our data needs and then optimized them through aggregation, giving us much faster response times on our requests.

Designing the backend API for proper functionality is crucial to our application's functionality. Controllers for each of the main collections (users, butterflies, species, and domains) were created with unique endpoints for the needed functionality. API endpoints were generalized and reused whenever possible to keep the complexity of the backend to a minimum. All request endpoints were tested in Bruno before being introduced into development on the frontend.

Creating the website data views required us to implement a user hierarchy within our UI where the admin has access to all of their facilities' data. We implemented a controller in the backend for the frontend to communicate for all data report types. The dynamicness of this communication was complex, but efficient. Upon making these requests, we implemented response handling to correctly display data in an orderly fashion. This view allows for data filtering and sorting for any of the user's needs.

Our first prototype was a baseline product that we presented to the client. This included a full guest user functionality with the front and backend linked and the application hosted on a live server. This demonstration allowed us to get feedback on design and functionality from our client and gave us a good starting point when resuming the project in the following semester. From this prototype, we iteratively improved on the functionality and continued to demonstrate new functionality to the client.

Adding other facilities to our application required us to create a dynamically adjusting application design based on the tagging system set up by the facility admin. When setting up a facility, the admin has the option of adding foreground colors, background colors, background shape, type of input characters allowed, and dots to their tagging parameters. This implementation covered the majority of cases for tagging systems. We also added the ability for admins to create authenticated accounts under their facility for trusted individuals. Authenticated admins of the facility can then register and access butterflies unique to their facility.

3.4.2. Gantt Chart Tasks (Second Semester)

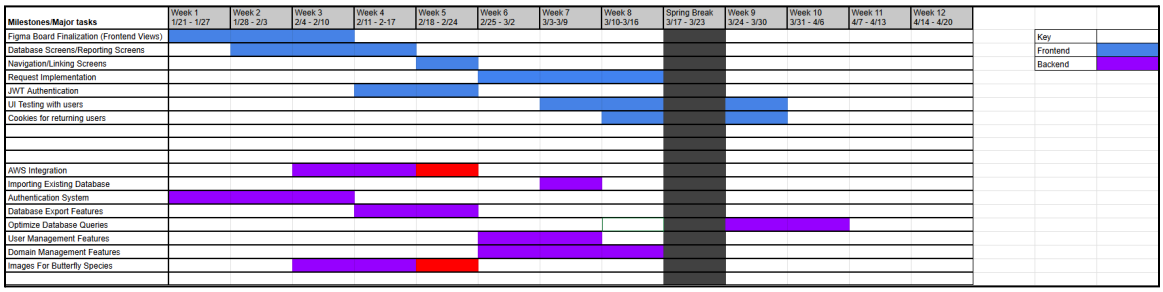


Figure 3.4.2 (Semester 2 Gantt Chart)

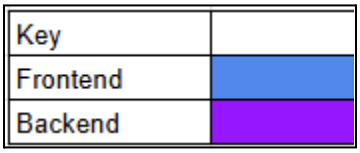


Figure 3.4.3 (Semester 2 Gantt Chart Key)

Frontend

Figma Board Finalization was our first priority when starting the semester. This included refining all exported screens from the provided Figma board to include the proper styling for desktop and mobile devices, along with the correct UI functionality for users. This was a requirement before we started implementing request functionality for individual screens.

Database Screens/Reporting Screens required us to implement a user hierarchy within our UI, where the admin has access to all of their facilities' data. We implemented a controller in the backend for the frontend to communicate for all data report types. The dynamicness of this communication was complex, but efficient. Upon making these requests, we implemented response handling to correctly display data in an orderly fashion. This view allows for data filtering and sorting for any of the user's needs.

Navigation and screen linking not only include page routing, but also variable management with our user hierarchy. Implementation includes functional navigation to all pages as well as storage of required information on the frontend throughout all pages for any user. Using this, we created an easy-to-understand flow of the web application for any user.

Custom request implementation for individual pages was needed for proper end-user functionality. This included mostly administrator features that needed more than just simple GET requests. Customizing the user experience was also impacted by

the backend request implementation for unique user login requests. Most of our existing endpoints got additional functionality for catering to multiple/facilities.

JWT Authentication required us to securely store and utilize the JWT token returned to the user upon logging in to their authenticated account or entering the website as a guest. JWT tokens were added in each request header to ensure security in the frontend.

Iteratively testing the UI with users helps us deliver a more refined final product. We have continuously updated deliverables for our client to constantly provide us feedback on design and functionality. As the project has reached a more polished final deliverable, the application has been distributed to more facilities for additional testing and feedback.

Backend

AWS Integration was a key part of our project's final deliverable. Our client was already using a personal AWS account for multiple services that the Reiman Garden is hosting. Gaining access and configuring our application to be deployed on an instance that is administered by our client is a core functionality we implemented for the best end-user experience. The AWS was also integrated with our GitLab CI/CD pipeline for automatic deployment of both the frontend and backend.

One of the core features our client requested was **importing the previous database information** from the current application they were using. This should fit seamlessly into our application hierarchy and work with all our features. Importing was done by exporting the current database information into a JSON document and then individually importing each of the sighting entries into our new database schema.

Security was absent in the previous implementation of this project. We chose to implement **JWT token authentication** for our authenticated users. This added authentication principles to each of our API endpoints to ensure no malicious users had access to the backend API. JWT authentication is done on the landing/login page for all users and is integrated properly with the guest user experience to ensure their functionality remains the same without needing an authenticated account.

Database exporting was simply implemented, where whatever data is currently being displayed to the authenticated admin user will be exported to a CSV file and downloaded to the user's device. This created an easy, streamlined operation to export generated reports on data in order to make graphical items to display the data.

Optimizing database queries was a lengthy task that required a lot of trial and error. We first created baseline queries for our data needs and then optimized them through

aggregation, giving us much faster response times on our requests. This helped us to meet the performance requirements given to us by our client.

For **user management features,** we implemented the ability for users to change their passwords as well as create their own unique username that will be associated with all of the sighting data they entered.

Domain management features required us to implement extra permissions for facility admins such as: creating and deleting users, registering new butterflies, marking dead butterflies, generating reports on data specific to their facility, and monitoring sighting data for incorrect sighting data.

Images for butterfly species were stored in an external, digital ocean database created by a previous group. We used a base URL that allowed base access to their images and simply appended the butterfly species to the end of the URL + .jpg to access the correct image. Any new species that is added to our system must also have an image added to the other project that is still being used for a different purpose by the facility, otherwise, a default image of a butterfly is used.

3.4.3. Deliverable Dates

Week 4: Initial Concept and Design Review

Week 8: Present responsive screens to the client

Week 15: Initial prototype with minimal functionality

Week 22: Second prototype with improvements based on client feedback

Week 26: Third prototype with improvements based on client feedback

Week 31: Finalized product

3.5 Risks and Risk Management/Mitigation

1) **Convert Figma Board to HTML**
    a) **Risk:** Views may not easily convert from Figma to HTML.
        i) **Probability:** 100%
            (1) We were already experiencing these issues and had seen this issue in the past.

ii) **Mitigation Strategy:** Begin conversion early in the project to allow sufficient time and resources for completion.

iii) **Result:** We did encounter problems while converting the Figma board screens, but were able to navigate through it. Lots of duplicate code was auto-generated and had us doing lots of cleanup. This slowed down progress and pushed back milestones due to us sticking with the plan of using the Figma board.

b) **Risk:** Views may not be fully responsive or accessible as planned.

i) **Probability:** 100%

(1) The exported views from Figma have never been fully responsive in our experience. This could be due to poor Figma design.

ii) **Mitigation Strategy:** Focus on achieving WCAG AA accessibility standards and conduct regular checks to ensure designs are adaptable to target screen sizes.

iii) **Result:** The views were not fully responsive when we exported them, as expected. However, we were able to alter the CSS of the views to become fully responsive to all device sizes.

2) **Create Database Objects**

a) **Risk:** Initial database objects may lack sufficient fields or functionality to meet project needs.

i) **Probability:** 75%

(1) We are unsure if we will need to restructure the data or add more fields to our data objects. We decided that from past projects, it is likely that we will need to change the structure of database objects in some capacity.

ii) **Mitigation Strategy:** build out a thorough list of all data points that need to be stored and get each of those data points with their constraints to be officially signed off by our client.

iii) **Result:** Our initial objects lacked fields that enabled full functionality of the application. We were able to update objects through team discussions and decide on the most efficient ways to update our database objects.

b) **Risk:** Containerizing the app may be more complex than expected.

i) **Probability:** 50%

(1) We have never containerized an application before, so we are unsure of the exact complexity of this task. We set our expectations high for complexity and think it could go either way of being more or less complex in actuality.

ii) **Mitigation Strategy:** Conduct early research to verify the compatibility of components and ensure they integrate smoothly into a container.

iii) **Result:** We ended up being able to do this relatively easily, and this was not an issue for us. We went through tutorials prior to containerizing the application in order to make this happen.

3) **Database Query System / Map backend to the database**
   a) **Risk:** Database performance may not meet client requirements.
      i) **Probability:** 80%
         (1) Our client is looking for an extremely fast database and data generation system, which we think may not be possible due to budget constraints for our resources.
      ii) **Mitigation Strategy:** Optimize the database structure and create indexes for high-demand queries. Track other potential optimizations during database creation to maintain performance.
      iii) **Result:** This was a problem early on in our project; however, through much optimization and testing of ways to query the database, we have met our client's performance needs.
4) **Multi-Page Navigation**
   a) **Risk:** Pages may not be easily navigable.
      i) **Probability:** 60%
         (1) As a developer, the site navigation will make sense to you because you made it. However, in past experiences, we know that it is slightly more likely than not that the navigation will not be as intuitive as we first thought, based on user feedback.
      ii) **Mitigation Strategy:** Test the navigation design with potential users to verify ease of use and page hierarchy effectiveness.
      iii) **Result:** This was a minor issue in our initial user testing, but through feedback, we were able to make the page navigation intuitive for our users.
5) **Butterfly Tagging System**
   a) **Risk:** The System may not support all common butterfly tagging methods used across sites.
      i) **Probability:** 70%
         (1) It is likely that an exhibit will use a tagging system that no one thought of, and we will not be able to account for it. Although for common tagging systems, we think we can cover most of them.
      ii) **Mitigation Strategy:** Consult the client and potential site owners about their tagging methods and ensure the system can incorporate all identified methods.
      iii) **Result:** Early on, our solution was only able to cover about 70% of tagging systems. However, we were able to design a system that allowed for almost any tagging system that you can think of for a butterfly. We

are confident that we have covered 99% of cases according to our client's instructions on how facilities tag butterflies.

6) **Design the Backend API**
    a) **Risk:** Backend APIs may lack adequate security.
        i) **Probability:** 50%
            (1) Our original backend layout will not be heavily focused on security, so it is likely that we will need to add more security measures in later iterations of the backend APIs.
        ii) **Mitigation Strategy:** Stay aware of common security risks and implement safeguards, including minimizing stored user data and maintaining regular backups for data integrity.
        iii) **Result:** We initially designed our application without a focus on security or user authentication. This was addressed later in the development timeline, where we added user authentication and adequate security measures to our backend APIs.

    b) **Risk:** The Initial API list may not cover all required software functions.
        i) **Probability:** 100%
            (1) We have created the list of the original API functions that we will need, but from past experience, you will never get everything on the first try.
        ii) **Mitigation Strategy:** Design APIs with extensibility in mind, allowing the team to add new functionality as needed.
        iii) **Result:** We had to add APIs as the project went along. This is expected in all software projects, as you can not account for what you do not know. This was not an issue for us and is a normal part of development.

7) **Create Website Data Views**
    a) **Risk:** The System may struggle to display complex data views efficiently, potentially affecting performance and data accuracy.
        i) **Probability:** 70%
            (1) The data will be complex, and we will likely struggle to efficiently query it due to the dynamic nature of the data.
        ii) **Mitigation Strategy:** Conduct performance testing for data-heavy views, consider pre-aggregating data to reduce load, and implement pagination for large datasets. Gather user feedback early to improve clarity and usability.
        iii) **Result:** Displaying the butterfly tags turned out to be a major issue and something that we spent a lot of time on. We decided to go with a solution that displays the tags with their colors and alpha codes all together as it would appear on the butterfly for simplicity. We found this design to be the optimal solution for the problem.

8) **First prototype**
    a) **Risk:** The Initial prototype may not align with client expectations, resulting in delays due to rework.
        i) **Probability:** 80%
            (1) We expect to receive constructive feedback from our client along with more features he thinks of as he is testing our prototype.
        ii) **Mitigation Strategy:** Hold frequent, iterative feedback sessions with the client and conduct regular checkpoints to integrate feedback progressively, reducing the need for major adjustments.
        iii) **Result:** We had to make adjustments to the application based on the feedback we received, but it was nothing drastic. The changes we had to make mostly included quality-of-life changes for our client and the users.

3.6 Personnel Effort Requirements

Below is the projected amount of effort for each task and subtask, along with an explanation of why we projected the amount of effort needed to complete the task.

Task	Subtask	Projected Effort(Person-hours)	Explanation
Core HTML Development	Converting Figma boards to functional HTML	20	Figma board to HTML conversions are clunky and require a lot of rework to function properly.
	Multi-Page navigation	10	Properly linking the web pages requires meticulous iterations and testing to ensure all links work.
	Facility management pages	20	The pages to update a facility's web page will be complex and require many different features to function.
	User log-in pages	5	The sign-in pages themselves shouldn't be too complicated on the HTML side.
	Mobile device optimization	10	Ensuring that mobile devices have the same experience can take a lot of optimization and rework.
Frontend-to-Backend Interactions	User sign-in implementation and authentication	15	Ensuring the sign-in is secure and cannot be exploited is a delicate process requiring time.
	Butterfly tagging support	15	Implementing tag posting and spotting will require complex interaction with the backend.
	Graphical data views	20	Creating useful and streamlined displays will require advanced

			methods we have not yet explored.
	Butterfly data filtering and sorting	15	Ensuring that the filters and sorting methods are efficient and effective will require a deep analysis of the data and structure of the database.
	Unique butterfly tagging for each facility	15	Allowing facilities to utilize various tagging methods requires great consideration of the possible methods and how to allow for them.
Database collection and Layout	Create a Database Management Scheme	15	Creating a scheme that can effectively manage all of the data requires a lot of research and a deep understanding to meet the needs properly.
	Create database objects	15	The objects are essential to the structure of the database and may need to be reworked if not done correctly the first time.
	Containerization	15	Complex and requires proper implementation to improve performance.
	Map backend to the database	20	Involves setting up complex database connections to the backend, which can cause efficiency issues if not properly mapped.
	Design the backend API	20	Very important interfaces for the front end to interact with

			that must allow for scalability.
User Testing	Test the website with the facility operators	5	Distributing the website and gathering feedback from the employees of Reiman
	Test the website with guests and the general public	5	Gathering feedback from the public utilizing surveys

3.6.1 (Projected Effort Table)

Below is the actual amount of effort put into each task and subtask, along with an explanation of why there were differences.

Task	Subtask	Expected Effort (Person-hours)	Actual Effort (Person-hours)	Explanation of differences
Core HTML Development	Converting Figma boards to functional HTML	20	40	We knew the Figma board would not easily convert, but we had to rework every single page, which was not expected.
	Multi-Page navigation	10	10	We were correct in our prediction, and we did not run into any issues while working through site navigation.
	Facility management pages	20	10	This was much simpler than expected because we were able to take the facility set up page and rework the request to do an update instead.
	User log-in pages	5	5	The development of the login pages was simple as expected.
	Mobile device optimization	10	20	We had to rework every screen for mobile optimization, some more difficult than others. We did

				not expect to have to do every screen.
Frontend-to-Back end Interactions	User sign-in implementation and authentication	15	10	Despite no experience with JWT tokens, the process was simple and easily implemented into our design.
	Butterfly tagging support	15	15	Accurate prediction due to the complexity of making a dynamic view, accounting for all tagging systems.
	Graphical data views	20	40	This was much more complex than expected and caused us lots of issues when trying to efficiently display large amounts of data.
	Butterfly data filtering and sorting	15	30	We had a lot of query optimization that needed to be done in order to meet performance requirements.
	Unique butterfly tagging for each facility	15	25	This took longer than expected due to adjusting our database objects to account for all tagging systems.
Database collection and Layout	Create a Database Management Scheme	15	15	This took as long as expected to plan and set up our database management scheme.
	Create database objects	15	25	Adjustments of database objects had to be made as we implemented functionality into the application.
	Containerization	15	15	Expected amount of time, as we were aware of the complexity.

	Map backend to the database	20	35	Ran into lots of efficiency problems and had to rework many of the queries to meet performance requirements.
	Design the backend API	20	20	We expected this to be ongoing development, and the amount of effort was to be expected.
User Testing	Test the website with the facility operators	5	10	We spent more time with our client and users in terms of receiving feedback and having the application tested.
	Test the website with guests and the general public	5	5	Our application was being tested for much more than 5 hours, but feedback gathering was around the expected amount of effort.

3.6.2 (Actual Effort Table)

3.7 Other Resource Requirements

The main resource requirement for our project would be hosting the service for our client on AWS while using GoDaddy for DNS management. This hosting service will be used to configure and control the status of the web application and make it available and easy to use for the client. Other external resources being used are MongoDB and Java Spring for the backend and database implementation. Java Spring is a free service that we are utilizing to secure CRUD requests between the front end and the database. Next would be the implementation of the database using MongoDB, which is a non-relational collection-based database. Here, we will store collections for each facility and track their specific tagged butterflies. We are also required to call a previous team's image database in order to pull the images of all of the butterflies. Lastly, we are implementing Docker so our client can control all services from a single place. This includes the cloud hosting through AWS and the backend service

implementation through a VM. Docker is a service offered by AWS, so integration will be quick and easy.

4. Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	Our project will directly impact butterfly exhibit employees as they will be implementing it into their facilities' everyday use. The project can improve the employees' welfare directly by improving already existing processes and making everyday tasks more efficient.	Reduce data report generation time. Enhance data into useful statistics for facility owners.
Global, cultural, and social	Our project aims to adapt to butterfly curators across multiple facilities and meet their needs. Multiple features of the website have been created specifically to meet the needs of those who are experienced in the field and provide them with useful information.	Inaccurate data or information about butterflies could result in harmful changes to the butterflies' treatment routines. Creating a tagging system that can work for each individual facility to generate useful data on their butterflies.
Environmental	Our project can affect the population of butterflies by changing the way that different facilities care for them, by providing detailed information.	Decrease the number of butterflies in an atrium. Change the plants used in the atrium to maximize butterfly health, leading to longer living butterflies.
Economic	Our project can reduce costs to our client by increasing efficiency in already existing processes and improving care methods for the butterflies.	Improved tagging efficiency leads to less wasted time for employees. Reduced hosting costs of the application without sacrificing performance or availability.

Table 4.1.1 (Broader Context Table)

4.1.2 Prior Work/Solutions

Monarch Watch App: The closest existing application we could find to what we are currently developing. This app is used to track specifically monarch butterflies as they travel across the world. This is a paid service that allows people around the world to

collaborate to track the path of monarch butterflies as they migrate. For more about the app, see [1].

Pros:

1. Easy to use
2. Provides accurate migration information
3. Allows for multiple users to work together

Cons:

1. Requires payment
2. Does not provide detailed data
3. Only used by one organization to track one species
4. Built for long-distance use

Solar-powered radio tags: Solar-powered radio tags and RFID tag combinations are close to being applied to butterflies. Currently, a small group is testing this technology with butterflies in the wild. This would allow for the automation of the detection and tracking of butterflies without manual entry from visible tags. For information about these tags, see [2].

Pros:

1. Precise tracking ability
2. Automated tracking
3. Low weight of 0.06 grams

Cons:

1. Not currently available on the market
2. Expensive
3. Still untested and could have issues

Small RFID Tags: Currently, some of the smallest RFID tags could be small enough to use on butterflies. Unfortunately, these tags have a very small range and high costs. “Smaller tags have a shorter read range since they cannot capture as much energy from a reader antenna.” For more information about the size and range of small RFID tags, see [3]

Pros:

1. Automated tracking
2. Small enough to be used on most butterfly species

Cons:

1. Expensive
2. Low range

Previous Project: A previous project to make a similar system was created a few years ago by a different senior design group from computer science. This project has functioned as a baseline, but it has many underlying issues that plague the site and need to be addressed. The client has requested that an entirely new website be created to replace the old system.

Pros:

1. Free
2. Sticker method of tracking

Cons:

1. Hard to navigate
2. Long load times
3. Missing polish
4. Missing needed features

4.1.3 Technical Complexity

This type of application does not currently exist in the industry; facilities commonly use CSV files to store information that is entered manually. All reporting data is challenging to report on, and no cross-facility data is being shared or utilized. Our application addresses this need for any facility to have a standardized way of entering data and reporting on it, with the additional advantage of comparing data across facilities.

The system has three components similar to any other software web application. Complexity grows when implementing a custom frontend for any number of facilities that utilize the application. This needs to be adaptable and provide growth for when we are no longer developing. Creating a way that each facility can have its own UI, along with tagging logic, increases the complexity of the project. Having all the data stored in one location provides an easier way to report on data from multiple facilities, but calls for strong logic and reporting options for users due to tagging system implementations. Taking raw sighting data and displaying it dynamically to administrators and researchers to a point where they can derive conclusions is the end goal of the project.

This requires extensive frontend development experience to be able to create a quick and easy-to-use application that is also scalable for any facility. Providing adaptive new endpoints through quality dynamic code poses challenges on the backend, which will manage any number of facilities that use the application. Database expertise is required not only to design and implement a multi-facility database but also to query and display information in valuable ways. This project requires experience from an extensive list of software technologies and systems, whereas usually, a developer focuses specifically on a single expertise or technology.

4.2 Design Exploration

4.2.1 Design Decisions

Butterfly Longevity Project – Figma This board contains all UI designs and the connection between each screen. This is important because we need to ensure that the screens have all the necessary functionality for the client's needs. We also need to ensure that they are in locations that would make sense to our clients and the users. This is arguably the most important design aspect of our project as it directly affects user experience in every way. You can not recover from a bad UI design. We have presented this Figma board to our client and have gotten approval.

MongoDB Atlas: Cloud Document Database | MongoDB We decided to use MongoDB as our database, which is a very important decision for our web applications' performance. This decision was important because it will directly impact how long it takes to load and query data. For our data types and the amount of data our database will store, we will benefit most from using MongoDB. MongoDB is very flexible and great at handling a dataset that will continue to grow without losing performance, which is a main concern of ours.

Cloud Computing Services - Amazon Web Services (AWS) We decided to use AWS to host our web application, which is very important to our application's availability. AWS offers high availability (99.999%), making it extremely reliable for our client and for the other users. AWS also allows you to upgrade your plan at any time in order to account for a higher amount of traffic. This allows our product to scale and have no performance issues. It is also very cost-effective in comparison to other options and is a pay-as-you-go price model, meaning we can cancel the service at any time.

4.2.2 Ideation

To decide on what database to use, we first looked at all of our data that would need to be stored within the database. Next, we estimated which data sets would have

exponential growth and which would remain similar in size as time goes on. We also listed performance and scalability as two of our most important features in the database. We chose those two traits specifically because our client's previous product had struggled with that, and we were looking to improve on the previous design. Our five considerations and why we chose to use/chose not to use them are as follows:

MongoDB (Using):

Pros

1. Flexible data schema
    1.1. Allows for easy changes in data collections
2. Allows for scalability of data size without a tradeoff in performance
3. High-performance speeds
    3.1. Fast querying speeds
    3.2. Fast retrieval speeds
4. Very little to no cost

Cons

1. Does not support traditional SQL joins
    1.1. Can lead to limitations in querying
2. Potential redundant data
    2.1. In turn, increasing storage costs

MySQL (Not using):

Pros

1. High performance speeds
    1.1. Fast querying speeds
    1.2. Fast retrieval speeds
2. Built-in security features
3. Supports vertical scalability

Cons

1. Does not support horizontal scalability
    1.1. i.e., adding traits to a data type
2. Lack of schema flexibility
3. Limited JSON support

Oracle (Not using):

Pros

1. High performance speeds
    1.1. Fast querying speeds
    1.2. Fast retrieval speeds
2. Multi-platform support

    2.1. Available on Windows, Linux, Unix, and more
3. High scalability without performance tradeoffs

Cons

1. High costs
2. Resource intensive
    2.1. Significant demand of CPU
3. Built for enterprise applications

IBM DB2 (Not using):

Pros

1. High performance speeds
    a. Fast querying speeds
    b. Fast retrieval speeds
2. Supports XML and JSON formatted data types
3. High scalability without performance tradeoffs

Cons

1. High costs
2. Complex setup and features in order to make the most of the application
3. Low flexibility of data schemas

MariaDB (Not using):

Pros

1. Open source and free to use
2. Compatible with MySQL
3. Flexible storage engines

Cons

1. Lacks advanced features that can improve performance
2. Low performance with large data sets
3. Lacks professional support, and there are not a lot of resources out there on the product

4.2.3 Decision-Making and Trade-Off

We created a weighted decision matrix to make a final decision on which database we would utilize for the project. We prioritized the performance and scalability of the platform first, as the previous project struggled with performance issues and could not

handle a large amount of data. Flexibility and cost are also important factors because of the vast range of data that needs to be stored and the low budget needed to keep the project running.

Weighted Decision Matrix (Scores range 1-10)

Database	Performance (25%)	Scalability (25%)	Flexibility (20%)	Cost (20%)	Querying Support (10%)	Total Weighted Score
MongoDB	9	9	9	10	7	9
MySQL	8	6	5	10	8	7.3
Oracle	9	9	6	3	8	7.1
IBM DB2	9	9	4	4	8	6.9
MariaDB	7	5	6	10	7	6.9

Table 4.2.3 (Weighted Decision Matrix)

Based on the scores of the table, we decided **MongoDB** was the best fit for our project after receiving a score of 9. Some of the following factors affected our decision and how we scored the databases.

Scalability: Horizontal scalability in MongoDB is a large advantage when handling large amounts of data that will only grow over time. We expect the database could contain many years of butterfly data that could cause issues on other platforms.

Flexibility: MongoDB allows for a very flexible schema structure that suits our needs, allowing the database to adapt as the project evolves without requiring complete restructuring.

Cost: Since MongoDB is open-source, it offers a great cost advantage over most other options, such as Oracle and IBM DB2, which require licensing and resource costs.

Performance: MongoDB's fast querying and retrieval speeds address previous performance issues the previous group has experienced, making it an even stronger candidate for our use.

Retrospect: MongoDB was not the ideal database for us, and we would have benefited from using a relational database like MySQL. However, since we realized this so late into the project, we were not able to make adjustments to our database and had to

stick with MongoDB, which is still a viable solution. The reason that a relational database would have benefited us is mainly through our database querying performance. We had to do quite a bit of extra work in order to make MongoDB perform at the level expected from our client in terms of data retrieval.

4.3 Final Design

4.3.1 Overview

We designed a web application that is user-friendly, fast, and reliable. The system is designed to deliver a smooth experience for anyone accessing it through a browser, whether that be on their phone, laptop, computer, or other devices. Users will be able to scan a QR code provided by the butterfly exhibits to access their uniquely customized site.

For our frontend, we build the user interface using HTML, CSS, and JavaScript. These technologies give us an easy way to create responsive pages without the use of external frameworks. Our site is lightweight, custom-designed, and easy to navigate.

When it comes to the backend, this is where we handle all of the logic related to data management and communication with the database. We opted to use Java Spring, an established, powerful tool that helps us to manage all of our application requests to the database. This layer allows for efficient communication between the frontend and the database. However, for images of butterflies, we utilize an external database associated with a previous project that is located on DigitalOcean. This will ensure that there is no confusion on imaging, as the other application already has the butterfly images.

To test our backend APIs, we utilize a tool called Bruno that allows us to make mock requests. Bruno provides us with an easy way to test new APIs and their performance as we are developing and altering the backend and database.

Our database is powered by MongoDB, a flexible database that helps with the dynamicness of our data. It securely stores all data and ensures that only authenticated users can access it. MongoDB provides us with easy access to update data at any point and monitor data as we integrate legacy data with our current database.

In order to make the application accessible, we have used Amazon Web Services (AWS) to host the website. This ensures that our web application is online, secure, and can scale with demand. On top of this, we are using GoDaddy to manage our domain name, which is how people will find the website. This allows us to create a relevant domain name that is easy for guest users to find and associate with us.

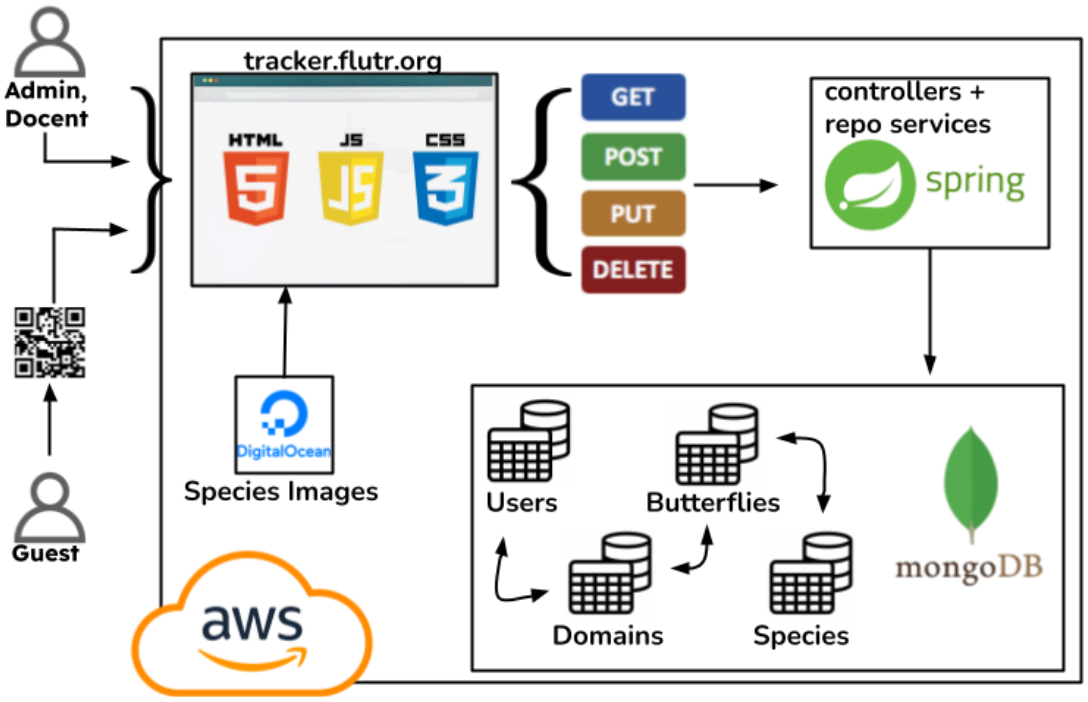


Figure 4.3.1 (Database Structure Diagram)

4.3.2 Detailed Design and Visual(s)

Frontend

The front-end implementation of the project is not using a framework such as React or Angular. Instead, it is written in pure HTML, CSS, and JS to mostly due to the Figma board code extraction we went through. All UX design follows the Butterfly Longevity Project Figma board, which can be found at the following link: Butterfly Longevity Project Figma.

PxCode is a tool used to generate HTML code from Figma boards. We utilized this to generate HTML code for each view that is presented on the Figma board. This leaves us with 14 generated views, so 14 generated HTML, CSS, and JS code files. A few views from the Figma board could not be easily containerized and were later manually created for the system. These views would include the database spreadsheet view and the admin home page setup view.

The code was then optimized to resize correctly to mobile, tablet, and desktop views of the web app. The main point of reference is through CSS files and styling guides, ensuring that top-level containers rely on viewport height and width rather than a pixel amount of some other variable identifier. Furthermore, low-level containers needed custom styling from developers to ensure proper function for all screen sizes.

Guest Experience: As defined before, a guest user is someone who is visiting the butterfly enclosure and will be entering butterfly sightings through a mobile device.

Upon creation of a facility, admins are given a unique URL for guests in the form of a QR code. QR codes are standard for what the Reiman Gardens have been using.

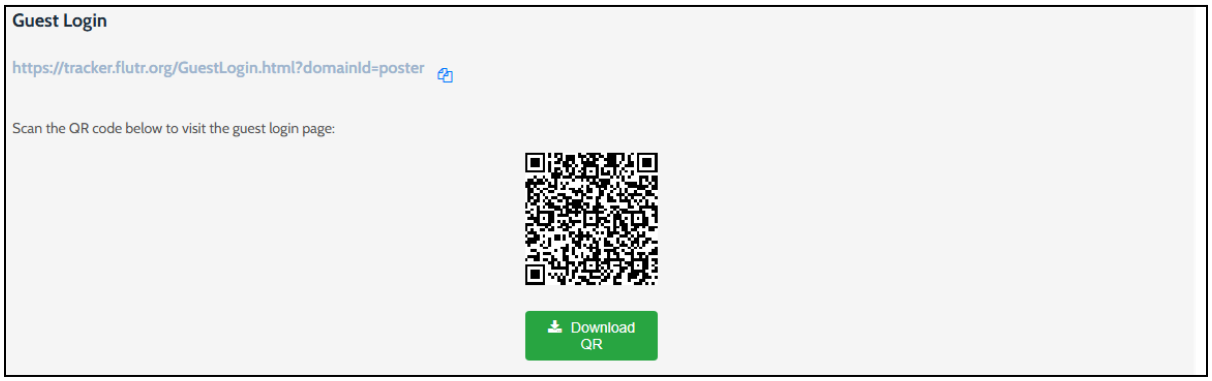


Figure 4.3.2-1 (Guest user URL and QR code on facility screen)

The coordinator who is letting guests into the butterfly enclosure will prompt all guests to scan the QR code and encourage them to utilize the application. Once a guest has entered the site, it will automatically be customized for the facility they are at per the unique URL they are given. The following flow is what a guest user will experience when utilizing the application.

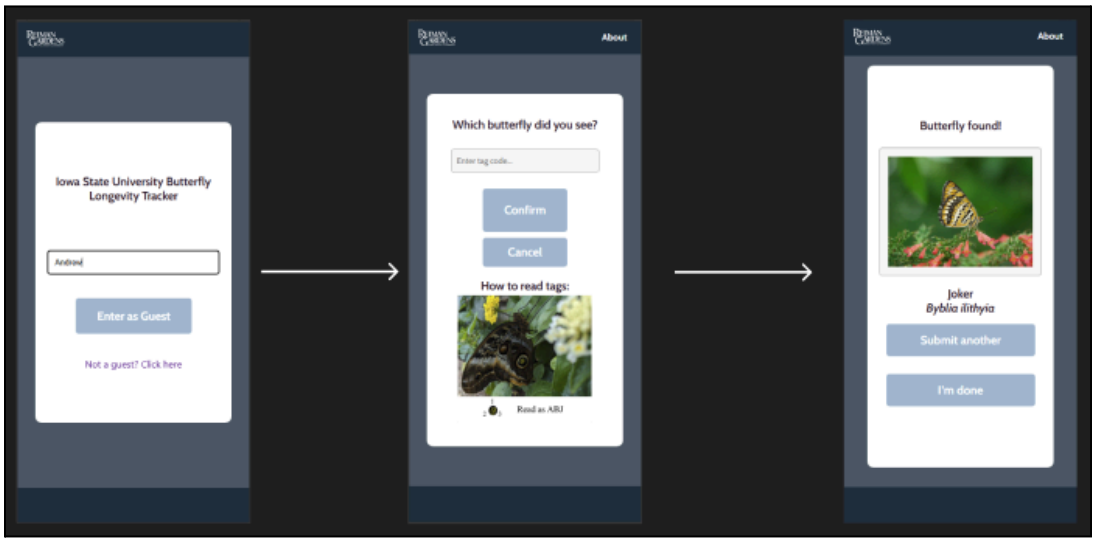


Figure 4.3.2-2 (Guest user page flow)

Users enter the application by providing their name, which is then linked to their user object, and they are identified as a guest. Guests are then prompted to enter a butterfly tag that they see within the enclosure. This page dynamically adjusts the tagging entry system to the facility they are at (changes entry functionality based on tag structure). The user is also prompted with a guide image that the facility selects upon creation to understand how to tag their specific butterflies.

Once a butterfly is tagged, the guest user will be shown a picture of the butterfly species along with some short information about the butterfly. Upon a successful sighting, the sighting is entered into the database for that facility.

Docent Experience

A Docent's experience is the same as a guest's experience, but rather, they will not log in with the QR code. Docents will have an authenticated account, which was made by the facility admin, to log in to. Docents then tag butterflies the same way as guests, and will be given some extra information about the specific butterfly they tagged (number of times sighted and how long the butterfly has been alive).

Admin Experience

An admin experience is far more complex than that of the guest experience. Admins will log in to their facility through an authenticated user account and be presented with a large amount of information.

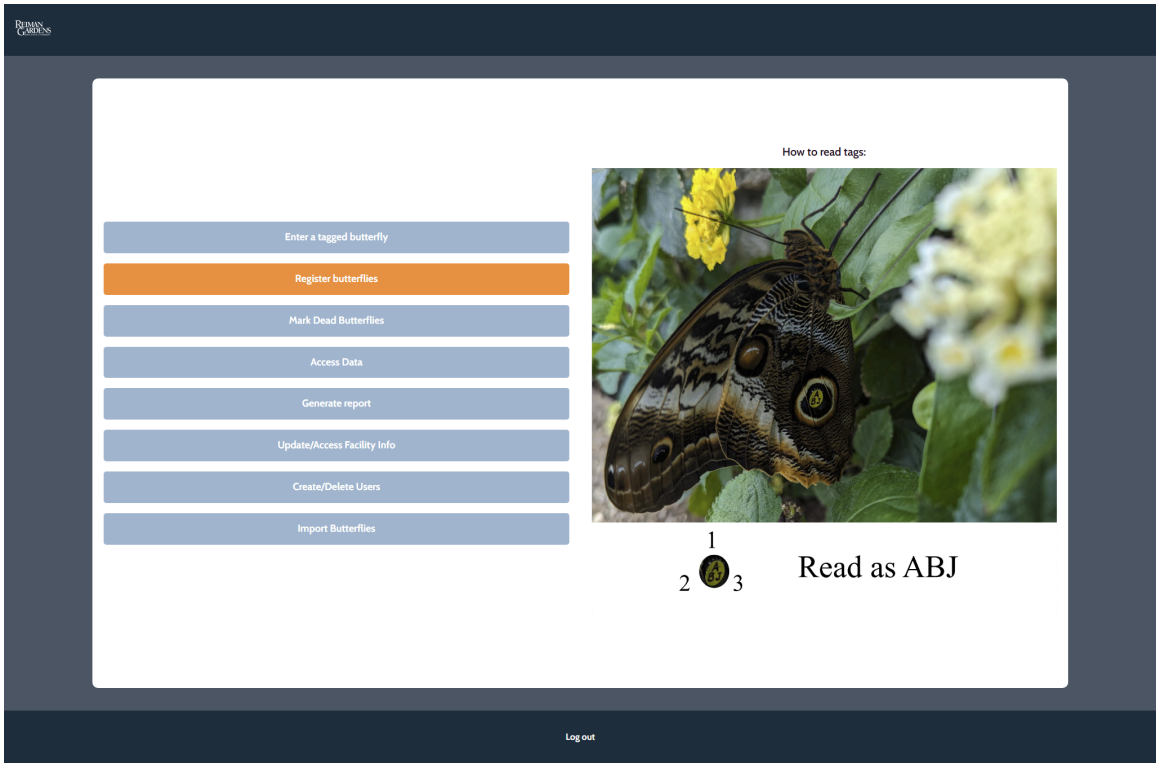


Figure 4.3.2-3 (Admin Home Page View)

From here, admins are able to control and configure their facilities application. They are able to enter sightings the same way a guest or docent would. They are able to register new butterflies by selecting a tag and species before releasing them into their enclosure. They are able to mark butterflies that have died within the enclosure, along with the data that they were found dead. They are able to access their facilities' sighting data, along with generating reports based on that data. They are able to update their facility information, which includes user images, logos, descriptions, and

more. They are able to create user accounts for their facility for either docents or other admin users. Lastly, they are able to import previous butterfly data into their facilities database.

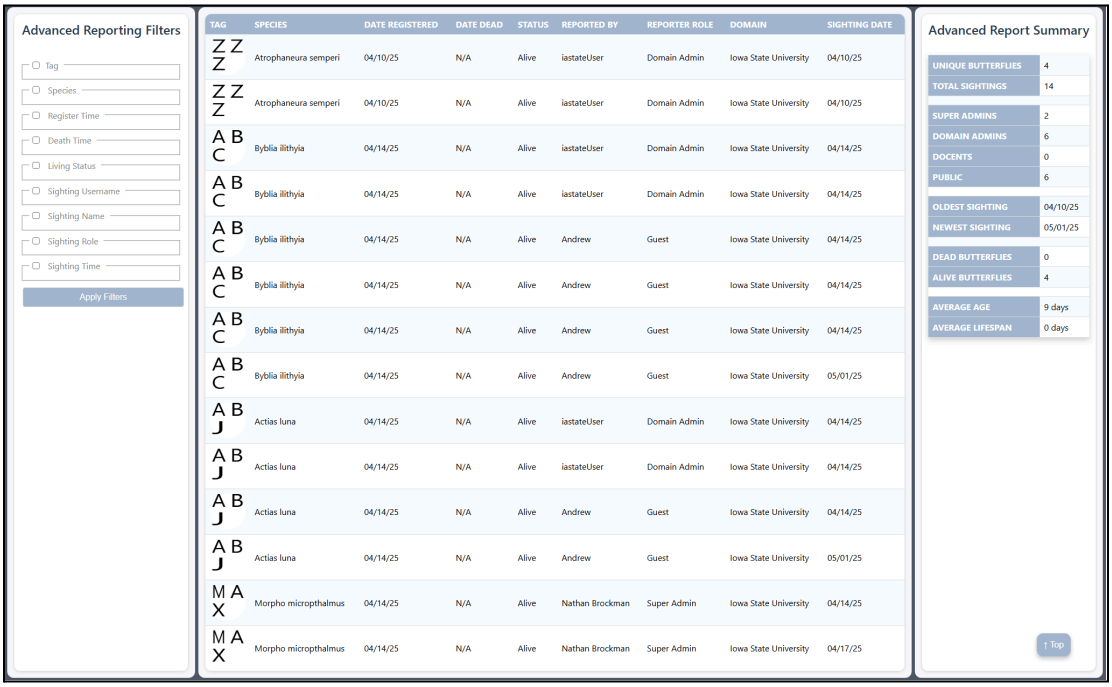


Figure 4.3.2-4 (Admin Access Data View)

The most important part of the Admin experience is viewing their sighting data. From here, they can apply filters to all sighting reports of any butterfly within their facility. They can also see statistics on their sightings based on the filters they select.

Note: Super Admin accounts are the same as regular admin accounts with slightly more control. Super Admins have the ability to create other facilities, as well as access data from *all* facilities that are using the web application. Reports have the additional functionality to filter based on facilities or lists of facilities. Super admins also have the ability to change butterfly species information for the whole web application.

See the video below (or the user operation guide) to fully understand the workflow for Admin accounts.

https://youtu.be/HrKjB3BT3NA

Backend

The backend is built using the Java Spring framework. The two main functions of the backend are to communicate data from the database to the frontend and to take in data from the frontend to store in the database. This is achieved by building a REST API with support for GET, POST, PUT, and DELETE HTTP requests. The Spring

framework allows for a simple implementation of additional services as well, such as authentication and database connection.

The Model classes in the backend define each of the main collections that are used in the database, along with any connections between collections. The Model classes will be created with respect to the client's requirements for what data needs to be stored. The Repository classes allow for definitions of queries on the database for a collection. The Repository classes are created to query information from the database that a user will need. The Controller classes are created based on the views of the frontend and what information they are required to have or give. The Controller classes allow for the HTTP requests to be made by the frontend to perform basic read and write operations on the database.

```
@GetMapping  no usages  Carter Awbrey +1
public ResponseEntity<Object> getAllButterflies(
        @AuthenticationPrincipal User currentUser) {
    if (currentUser.getRole() == BltRole.SUPER_ADMIN) {
        return ResponseEntity.ok(butterflyRepository.getAllButterflies().getMappedResults());
    }

    if (currentUser.getRole() == BltRole.DOMAIN_ADMIN || currentUser.getRole() == BltRole.DOCENT) {
        return ResponseEntity.ok(butterflyRepository.getAllButterfliesByDomain(
                currentUser.getDomain().getDomainId()).getMappedResults());
    }

    return ResponseEntity.status(HttpStatus.FORBIDDEN)
            .body(ControllerHelper.ErrorMap("Unable to query list as guest"));
}
```

Figure 4.3.2-5 (Butterfly Controller Method)

The following table entries map out all of the main API requests that have been implemented for each HTTP request type. The requests utilize JSON for communication (i.e., request and response type).

Get	Post	Put	Delete
Get Facility Theme	Login Request	Update Facility Theme	Delete Invalid Butterfly Sightings
Get Facility Assets (Logo, Tag Method, etc.)	Post Butterfly Sighting	Update Facility Assets	Delete User Accounts
Get Butterfly information for a specific facility	Create Facilities and Facility Admin Accounts	Update the Admin Account password or information	Delete Facility and Facility Information
Get all butterfly	Add a new	Update the	Delete/Remove

information with specified filters	Butterfly Species	butterfly sightings information	Butterfly Species

Table 4.3.2-6 (API Request Table)

Database

MongoDB was selected to be the database to store all the information in. It is a non-relational database, meaning that all information is stored in collections, which follow a JSON format. The data is stored in four collections: Butterflies, Species, Users, and Domains. The Butterflies collection refers to the butterflies that are registered to a butterfly exhibit (domain). The Species collection is a list of all valid butterfly species. The Domains collection refers to each individual butterfly exhibit that can house their own butterflies. The Users collection holds each individual user with what domain they belong to and which butterflies they have tagged.

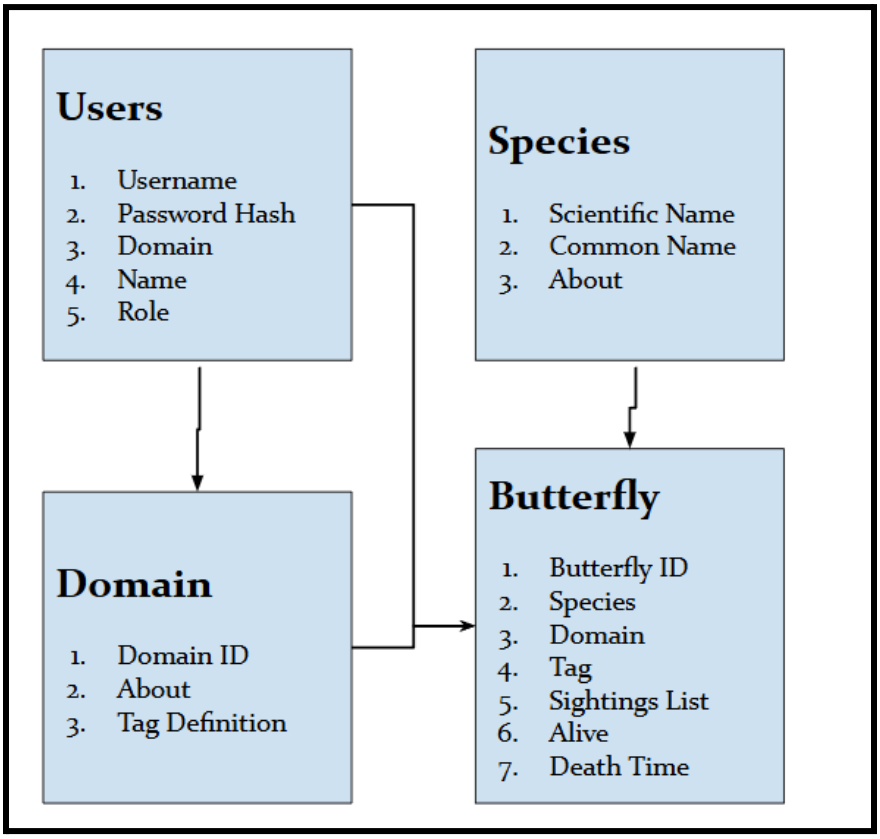


Figure 4.3.2-7 (Basic Database Schema)

Due to the possibility of increased complexity and cost, all data is stored within the four mentioned collections. Since all the data is stored together, a permission schema has been created that will define the access rules for users and their domains. A Domain_Admin can only access and modify information within their own domain. The Domain_Admin cannot access or modify any information that belongs to another

domain in any case. The Super_Admin will not have any location or data restrictions. They will have full access privileges to all data that is present in the database.

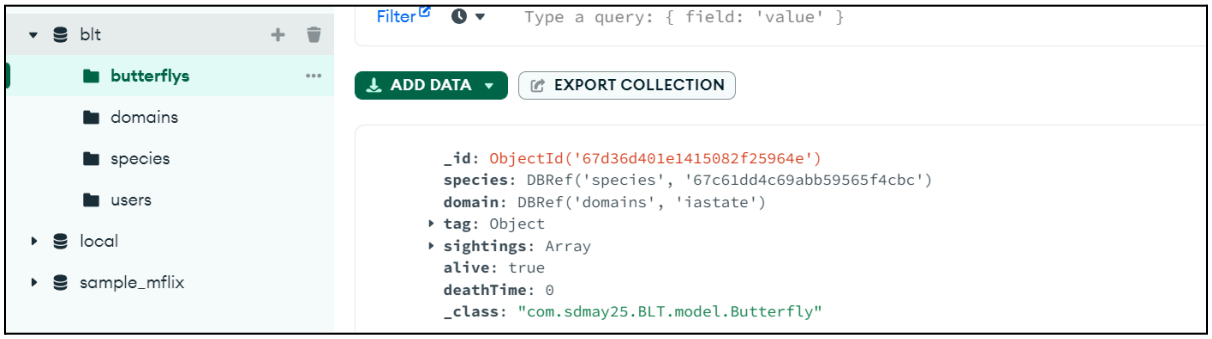


Figure 4.3.2-8 (Database Domain Example)

4.3.3 Functionality

Our design centers on two primary use cases: logging butterfly sightings into the database and outputting and analyzing this sighting data. The app lets users easily input butterfly tags, time, and other details depending on the user's role. These inputs are stored in a database with a unique identifier for the butterfly exhibit for streamlined analysis and retrieval.

Additionally, the app includes a robust user management system, allowing for varying access levels based on user-authenticated roles. This ensures that different users, from researchers to citizen scientists, have tailored access to the database and app features. For example, while a facility administrator may access detailed data and analytical tools, a casual user may only input sightings.

The design is also adaptable across institutions, enabling each to manage its own data access policies and customize the user experience as needed. This flexibility ensures the app serves diverse institutional needs, facilitating collaborative and secure data collection across multiple organizations.

4.3.4 Areas of Challenge

Converting from Figma

The process of converting views from the Figma board to actual functioning HTML was a lot more difficult than we initially expected. We ended up having to modify almost all of the generated code, and many changes were needed to get the sites working properly. This was partially due to the way the Figma board was created without consideration for HTML development, making many of the groupings and elements entirely out of place, requiring a lot of TLC.

Data efficiency

Making sure that all functions of the website that involve data calls meet our response time requirements required a lot more time than we expected. Initial methods for grabbing data were often very inefficient and had to be reworked on both the front and back ends in order to bring load times down. This was challenging and required learning more about how to efficiently work with large data sets while keeping the interface user-friendly.

Database to Frontend formatting

When creating interactions between the frontend and backend, it often requires a lot of discussion to reach the best solution for data formatting. This is due to the fact that many of the backend functionalities require very detailed and precise information for their in-depth data structures, while the frontend would be taking in base-level user input most of the time. We often had to discuss and find a middle ground where both sides could function efficiently in order to meet the requirements of both sides.

4.4 Technology Considerations

MongoDB: This database is ideal for flexible, semi-structured data like sightings. While it lacks some transactional integrity compared to SQL databases, its scalability and JSON-friendly structure make it a strong choice. Although a SQL database like PostgreSQL could improve data consistency, MongoDB's flexibility aligns better with our needs.

Spring Boot: Chosen for its robustness and strong support for RESTful APIs, Spring Boot simplifies backend development and integrates seamlessly with MongoDB. Although complex, its built-in features reduce boilerplate, saving time. Alternatives like Node.js could offer a lighter stack, but Spring Boot's stability and Java-based environment are ideal for our goals.

HTML/CSS/JavaScript: These core web technologies provide broad compatibility and control over the UI, allowing us to build a responsive and accessible interface. While frameworks like React or Vue.js could streamline development, using plain HTML/CSS/JavaScript keeps our front end lightweight and manageable.

5 Testing

5.1 Unit Testing

Frontend units being tested:

In order to test our frontend UI components, we manually viewed the page in separate browsers (Chrome, Edge, Safari) to ensure compatibility. In each of those browsers, we checked the responsiveness of each page to different screen sizes by adjusting the screen size of the browser. This helped us to ensure compatibility with different device types and sizes.

When testing our communication with the backend APIs, we first used Bruno to understand the request and how it works. Then, once we have verified the functionality, we implemented the request in the frontend and added proper error handling for edge cases and for unforeseen response types.

To test our page navigation, we manually did end-to-end testing, ensuring that every page has a navigating path to any other page.

Backend/Database units are being tested using JUnit tests:

There are several features that need to be tested for their basic functionality. The tagging system that was created has several basic features that have been tested. The tag system needs to ensure that tags can identify butterflies and that tags can correctly belong to their tagging system. The tag definitions need to properly validate tags to ensure that the tag falls under the rules of the tagging system. This also includes pre-generated tags that also fall under the tagging system. When comparing tags and tag systems, ensure that the hash coding system is able to properly compare the tags.

Butterflies need to be tested so that duplicate butterflies cannot be entered into the database unless the duplicate butterflies are marked as dead. When inserting or deleting data from the database, the insertion and deletion processes need to ensure that they are only deleting or inserting what is expected. Permissions will need to be enforced during the regular API calls. The permissions must follow the outlined permission schema that our client has provided. The permission scheme will not allow for any domains to access each other's data.

5.2 Interface Testing

Frontend to Backend:

Interface face testing for the frontend was done mostly through the testing of API calls. All API call testing was done through Bruno. When new requests were implemented, a Bruno request was made for frontend members to use and test the request while in development. While testing during development, this allows members to understand request and response bodies for both proper and improper requests to improve error handling robustness.

All requests were tested with their respective end points, which include all GET, POST, PUT, and DELETE requests.

Backend to Database:

Interface testing for the backend was mostly done through query validation. Validating query data flow, results, and result formatting for all requests was done through the implementation of requests in Bruno. Incoming request bodies and response bodies are easily validated through the testing of the request through Bruno.

Response times of requests were also a big point of testing with the backend implementation. When it comes to accessing large sets of data for reporting information, many tests were administered to optimize the run time when retrieving large sets of data. Most of this was done through Bruno, but small amounts of testing were done on the live server to confirm run times within the AWS VM instance (changes in compute power have impacts on query run time). Optimization was then done through aggregate creation on the backend to specify how sets of data should be handled within the database when it comes to foreign keys (DB references).

5.3 Integration Testing

Backend Integration:

Integration tests involve running a MongoDB instance (or an embedded database) to verify actual database operations. Once the database and backend are connected, manual checks are performed to validate key workflows like user creation, tag assignment, and data retrieval for sightings.

Backend-Frontend Integration:

Once the frontend and backend are connected, we manually test the data-driven functionalities to ensure they work as intended. This includes workflows such as: User authentication and authorization, tagging and spotting processes, and displaying data for various institutions and users. Since the API is already verified during backend testing, the focus is on ensuring the integration works as expected in real-world scenarios. This approach ensures that any issues arising from frontend-backend interactions are identified and resolved efficiently without duplicating API-level verifications.

Screen Functionality Integration for Frontend:

Navigational Integration involves ensuring that the new page can be properly navigated to and from based on existing screens. After the development of new pages with functionality for navigation, the pages would be tested to ensure the correct functionality. When using GitLab for our workflow, it was simple to merge new pages into the development or production branches to then test with other page navigation.

Data Fetching Integration ensures the page properly receives and displays all data obtained from the request for the current user. Testing for this lied discreetly in

development, as we utilized local and session storage of user or facility information to be reused on any page the user navigated to, rather than making new requests for the same information. As a result, data overlap within the frontend of the system was kept to a minimum.

5.4 System Testing

For the frontend, we utilized a live development environment to test changes through manual end-to-end testing. This live development environment allowed us to simulate our production environment, ensuring that functionality would be consistent when we deployed our web application. This created a smooth deployment process for our frontend application onto the live production server.

Our system-level testing strategy ensures the backend operates reliably by combining automated and manual approaches. Unit tests verify individual components, ensuring their functionality in isolation. Interface and integration tests use a hosted MongoDB instance to validate operations like inserting, retrieving, and deleting data while confirming relationships and constraints are upheld.

For deployment, we perform manual testing to ensure services are accessible externally and workflows like user registration and tag assignment function as expected. This layered approach ensures all components and their interactions meet project requirements while focusing on practical, real-world functionality.

5.5 Regression Testing

We set up automated test cases that ran each time we pushed new code to one of our branches on GitLab, providing us with automated integration testing. We created a CI/CD pipeline within GitLab to help us achieve this. Because we were using git, we were able to utilize the version control features to revert to old changes if an updated code version did not provide the results we expected. Each branch was run against the CI/CD test cases prior to merging, meaning that all code merged had been tested with the current working version of the application.

5.6 Acceptance Testing

We ensured that all performance metrics were met, which includes page load time and request response time. We also confirmed that exported files of data reports are of a high quality and manageable download size. Along with these quantitative acceptance tests, we also had regular qualitative testing in the form of direct meetings with the client/product owner to ensure project scope and goals are met at all times throughout the process. These qualitative tests, along with strong communication with the client, allowed us to meet the needs of our client.

5.7 User Testing

For our user testing, we delivered prototypes to our client that were distributed to other butterfly facilities and used for a short amount of time to test functionality and user interaction for guests, docents, and admin user types. In order to set up the facilities and navigate the application, we first made and sent a manual to our client. After our client and the other facilities set up their customized domain, they were able to give access to the guests to test out the report features of tagging butterflies. The guest users were able to participate by visiting the butterfly facility and scanning the QR code that led them to the facility's website. Then, the guest users were able to enter the tags of butterflies that they spotted around the exhibit. This data was then stored in the database, and the admins of each facility were able to generate reports and access the generated data.

From our user testing phases, we got lots of valuable feedback on improvements we could make in order to improve the user experience for all user types. The overall general reaction to the application from our users was very positive. However, there were lots of little changes that we made in order to enhance the experience of the users and to tailor to the needs of the administrators in terms of data displaying. Something we noticed in our first prototype was that the admin users were not able to intuitively navigate through the application. We communicated with the administrators on what they would like to see change for the navigation and made the necessary changes for our next prototype. The next time we delivered a prototype to the administrators, we were able to see a large difference in their ability to navigate the application.

5.8 Security Testing

Since our web app was marketed to other facilities for usage, information privacy was a strong concern. Sensitive information for each facility involved was held securely in the database. We took standard security principles into account when developing both the frontend and backend of the application in order to provide our users with a secure experience.

Since we used AWS and Docker to host both the database and the frontend web app, we have a strong built-in security infrastructure in place. We focused primarily on implementing high security features on the backend to ensure that it can not be manipulated in any way. We prevented query injection into fields from an attacker attempting to retrieve sensitive database information pertaining to facility or user account data. Along with this, we have ensured that each request was properly formatted and cannot have request parameters nor request bodies manipulated by an outside user trying to break the site. This was implemented by both frontend and backend logic to ensure user input is treated properly by the system.

Vulnerability testing was one of the later forms of security testing we do. As stated before, AWS has a strong infrastructure in terms of machine security, meaning we had little work to do when configuring host machine security; however, vulnerability testing, port scanning, and more were administered to ensure safety. This was also adapted with AWS machine configuration to ensure ports were closed, secure versions of operating systems were chosen, and no severe connection vulnerabilities were present in which an attacker could gain access and enumerate.

Lastly, since we used AWS, we have longevity with machine security as AWS security systems will be updated even after we have completed development on the project.

5.9 Results

Our testing results have indicated strong compliance with requirements and user needs. All unit tests have passed successfully, validating the individual components of our system. Additionally, manual testing of workflows and components confirms that the design performs as expected. While these tests do not guarantee an error-free implementation, they significantly reduce the likelihood of bugs in the software.

Qualitatively, client feedback has been positive, particularly regarding the UI design. The service's speed meets user expectations, and additional optimization is deemed unnecessary at this stage due to the already satisfactory user experience. Moving forward, our focus will remain on maintaining this level of reliability as new features are integrated.

6 Implementation

We have implemented a fully functional website that allows users to easily and efficiently sight and track butterflies throughout multiple facilities. This involves allowing guests to enter sightings of butterflies at any facility they may be visiting, while also giving facility workers the ability to perform much more advanced tasks in order to manage their site. This involves adding new butterflies, viewing report data based on sightings, managing and setting up new facilities, deleting unwanted entries, and more. Our goal when beginning this project was to provide the users with a full suite of tools for the efficient and quick management of site butterfly tracking without needing a background in database management. Our tracking solution allows them to view important data about their butterflies' release dates, species, totals, and even lifespans in order to give them useful information on how they are performing compared to other sites and what may be working or not working.

One feature that was unable to be implemented was importing previous databases for any facility. We were able to successfully add an import feature for putting previous data they may have collected into our new system. But, this requires the data to be structured in a very specific way that many facilities may not have previously used. This means that case-by-case data will have to be restructured in a unique way in order for it to align with the structure of our database, making it a very difficult task to create a universal solution that is simple enough for anyone to use. We have provided a method if some time is spent reformatting data, but it may not be perfect for some different scenarios.

6.1 Design Analysis

We have ensured through personal and user testing that all of our pages are responsive and perform the needed function without error in order to deliver a streamlined user experience. Our primary goal was to make sighting a butterfly as simple and intuitive as possible for a user, and this has been achieved by utilizing user feedback and iterating upon previous designs. We had to ensure that we did not overcomplicate any tasks while still having them be in-depth enough to efficiently perform the needed tasks. This works well as our target user group often cannot have a strong background in technology and needs the site to be as easy to learn as possible.

Our main area that does not work as expected is the data importing. It is not very user-friendly compared to the rest of the site, as it requires an understanding of data structures like JSON and how to convert large data into a specified format. This is a time-consuming and difficult task that can be very error-prone if you are unfamiliar with the process. Also, importing all of this previous data can take quite some time if it is a very large dataset. In order to make this a smoother process, we could have allowed

our database to be more adaptable and one-size-fits-all. But, this would also lead to worse performance and possibly make it more error-prone in the future.

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	Definition	ACM Code of Ethics	Team Description
Work Competence	Work is completed to the highest quality within our team's ability	2.1 Strive to achieve high quality in both the processes and products of professional work	Regular code testing and coding reviews prior to merging changes to our main branches
Financial Responsibility	Keep costs to a minimum while maintaining quality	1.2 Avoid harm	Researched several hosting services and picked the one that is most cost-effective
Communication Honesty	Give updates on work you have done, plan to do, and are currently doing	1.3 Be honest and trustworthy	Sharing with each other what we are working on
Health, Safety, Well-Being	Ensure the security of the facilities and users	1.2 Avoid harm	We are looking into implementing security for our website
Property Ownership	Respect the ownership of resources and ideas available to us	1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts	We ensure that all of the content we use is either cited or free for use
Sustainability	Ensure environmentally friendly software and use sustainable tools	1.1 Contribute to society and to human well-being, acknowledging	We chose technologies that are well-established and will be

		that all people are stakeholders in computing.	serviced for many years to come
Social Responsibility	Upkeep user engagement and involvement amongst guests at the facilities	2.7 Foster public awareness and understanding of computing, related technologies, and their consequences.	We designed our UI in a way to keep our users engaged

Table 7.1 (Areas of Professional Responsibility/Codes of Ethics table)

What we did:

Throughout the development of the project, we kept regular contact via Discord messaging to update each other on the work that we are doing. We used Git and GitLab for version control as well as quality control. We utilized code reviews and code testing prior to merging any code changes to our main branches. This ensures that we keep a high quality of code and reduce the risk of finding bugs down the line. We saw our progress through our GitLab issue board, which contained all of the things we planned on doing, were currently working on, and had finished working on. We implemented existing software features in order to improve our design and increase sustainability.

Area we performed well in:

Work Competence: We ensured that all code changes were reviewed and tested by at least one other member of the team before merging a branch. This ensured quality work and kept us all accountable for fixing any oversights in our own code caught by another team member.

Area we could have improved:

Communication Honesty: We did not always keep each other updated on what we were working on, and there were disconnects between what we expected someone to be doing vs what they were actually doing. This was harmful to our progress, as we wouldn't want to assume something is being done when it is actually not being worked on. We could have had more frequent and formal team meetings to prevent this, rather than primarily communicating through Discord via text.

7.2 Four Principles

	Beneficence (Promoting Good)	Nonmaleficence (Avoiding Harm)	Respect for Autonomy	Justice
Public health, safety, and welfare	Our project design greatly increases the performance and ability of administrators to report, which affects their day-to-day workflow.	Our design does not harm the general well-being of users in any way.	Our design allows each facility to configure its UI in its own way, which can benefit its personal brand and image.	Our design will have the same functionality for each facility, not creating an imbalance of features from location to location.
Global, cultural, and social	Our design promotes a more accessible and well-performing workplace within any facility that decides to utilize our application.	Our design does not harm the values of any specific cultural groups or people,	Since our design isn't required to be used by guests, there is no cultural autonomy impact on our design.	Our project design will benefit any facility enclosure workspace for administrators, volunteers, and guests in a multitude of ways.
Environmental	The design does aim to provide data to administrators and researchers to benefit the longevity of butterflies, which I would say is a good thing to promote.	Our design does depend on tagged butterfly enclosures which could be seen as a harm to the natural environment for butterflies.	Tagging butterflies could have an environmental impact, but our project builds on the basis that the butterflies are already tagged. However, this is likely still a negative impact on butterfly autonomy.	Our project design will provide butterfly enclosure administrators with relevant information on butterfly life spans based on species. This can be used to derive the environmental impacts of enclosures on

				specific species' lifespans.
Economic	Our web application will provide Butterfly Exhibits with an inexpensive way to maintain their data while keeping a high-quality user experience for their guests.	We chose an inexpensive hosting service in order to mitigate the amount of money our client needs to spend.	Exhibits don't have to pay for a new tagging system, as our web application will be able to handle any system they choose to use.	All exhibits will have access to the web application at no cost.

Table 7.2 (Four Principles Table)

7.3 Virtues

Team Virtues

Integrity: Integrity was crucial to our team both when it came to development and teamwork interaction. In terms of development, the integrity of code as well as the moral integrity of each team member was kept to a high standard. Integrity of team members when it comes to deadlines, functionality implementations (quality of work), and honesty when facing issues was crucial to our success.

Perseverance: A large part of being a software developer is being able to persevere through problems. Most software developers will tell you stories of debugging code for hours just to find that a single line of syntax was causing the whole system to not function properly. As a team, we found perseverance very important to being able to meet our goals. Valuing perseverance as a team helped lead us to better collaboration and ultimately a better end product.

Collaboration: When working on a large software project like the one we were faced with, collaboration is of utmost importance. Being able to collaborate effectively in our own way helped us create a high-quality product. Collaboration through our GitLab workflow benefited us greatly while also teaching us the proper ways of collaborating on a mutual goal in the industry. Each team member was able to contribute in their own way at their own pace for the mutual team goal through our team's collaboration.

Andrew

Accountability has been a virtue I have demonstrated thus far in our senior design project. This is important to my character and taking responsibility for my actions, along with taking responsibility for my assigned work for the team. Persevering and communicating with my team to ensure I meet deadlines and produce quality work is key when it comes to accountability. I have shown this virtue by being clear and honest with my team in my work progress and communicating my own responsibilities for the project.

Determination is a virtue that I would like to work on moving forward with the project. Although I think I am determined to meet deadlines and project goals, I haven't acted on this virtue up to this point. Being more determined in the future could help the team collaboratively push the project further than its intended goal and boost team performance. To demonstrate this virtue, I could take more of a leadership role and accept more accountability for the project in order to increase determination in finishing the project for not only me but my team as well.

Alex

Collaboration is a virtue that I have shown while working on this project through participating in code reviews and sharing my work and knowledge with other group members. I ensure that all of the code I add is tested and reviewed by another team member to align with our team's standards. I make sure that everyone is aware of what I am working on and make sure to update team members on the work we need to complete.

Resilience is a virtue that I could improve upon going forward with the project. I feel that sometimes I will run into a difficult problem, and it will deter me from completing work that I need to get done. I sometimes put the work on hold for too long, and it could delay my completion time when I could have completed it earlier.

Charlie

Reliability is a virtue that I have shown consistently throughout this project. It is important to always follow through with your word given to your teammates, especially when working with deadlines, as it can cause incomplete products if not properly addressed. I have shown this throughout the semester by ensuring that anything I commit to is complete by the needed deadlines.

Creativity is a virtue that can lead to great solutions for a problem. Unfortunately, I feel that sometimes I have not taken full advantage of this virtue and resort to the safe option of doing what I know already. This virtue is important for innovating and creating new, better solutions to existing problems. To demonstrate this virtue, I could

start to put more effort into thinking of new possible methods to address issues that I may have previously handled in a different way.

Carter

Tenacity has been key for me during this project. It's important to follow through on commitments, especially with deadlines, to avoid leaving tasks unfinished. I've made sure to meet every deadline and complete all my responsibilities on time throughout the semester.

Punctuality is crucial for keeping everything on track, but I recognize I've struggled with being on time for meetings and check-ins. Being punctual helps maintain momentum and respect for everyone's time. To improve this, I plan to better manage my schedule and set earlier reminders to ensure I'm always on time.

Jaret

Adaptability is a crucial skill and component that a software engineer must have. Many times, a project may need to be adjusted or an approach will need to be changed since there are flaws existing in the original plan. The ability to willingly adjust your plan and make the necessary changes is an important part of the job of a software engineer

Perseverance is a virtue that is important for a software engineer to possess. Software development is not always a pretty process and can bring many headaches to the user. This virtue has been demonstrated throughout the semester since there have been several small roadblocks and challenges that the team has faced and needed to overcome.

8 Conclusions

8.1 Summary of Progress

Throughout the course of this class, we were able to develop and complete a fully functional website for tracking butterflies within facilities across the nation. This involved implementing hundreds of features in both the backend and frontend in order to achieve the functionality needed. We have been able to function cohesively as a team to reach the results we wanted and are satisfied with our accomplishments. The website we have created is fully developed and allows facilities to retrieve detailed, precise information and statistics while also providing an interactive experience for visitors to learn more about the butterflies they are seeing. It also meets our responsiveness goals, allowing the website to be a very streamlined experience.

Testing of the application was done progressively throughout the development of the application. The most important testing was client satisfaction; over the development of the application, the client was always keyed in on UI or functionality of page designs to ensure compliance and satisfaction with the end product. The application is now being used by our client along with a number of other facilities for testing. The compliance of the application is actively being modified to meet user needs for all the facilities that are utilizing it in different ways. As we near the end, the number of modifications to the functionality of the application has come to a stop, as we have addressed the majority of user needs that required additional functionality to be incorporated.

8.2 Value Provided

The Global Butterfly Longevity Tracker aimed to fulfill a need for several different butterfly facilities across the United States. Right now, several butterfly facilities do not have a simple and free-to-use application that can track butterfly sightings and lifespan data. Some available solutions would include closed-source and paid software, provide the necessary data within a spreadsheet, or mark all the data down on paper manually. The latter two solutions do not allow for an easy experience to generate specialized reports based on information in the database. None of these solutions allows for features that can enhance the guests' experience when they are visiting a butterfly exhibit.

This project has been built to have a simple layout for the user and for an administrator at the butterfly exhibit. For the guest view, they will scan a QR code and be prompted with a box that asks them to enter the butterfly code. The administrator view will allow them to perform administrative actions on their domain, such as adding new butterflies, editing butterflies, generating reports, editing domain information, generating reports, and several more features. This application allows for butterfly

facilities to have a free alternative that can help them handle massive amounts of data about butterfly lifespans for their registered butterflies.

8.3 Next Steps

This project has met all the requirements our client initially laid out. Throughout development, new milestones for functional requirements were introduced and dynamically added into the final product of the application. The hosting of the application has been set up on the client's own AWS account and at a lower cost than the previous implementation. This gives the client full control over the application and when it will be live or not. The scope of our project requirements has all been delivered upon within our final iteration. From here, the web application will be distributed and used by numerous facilities similar to the Reiman Gardens. Each of the facilities will be able to communicate with our client for proper use of the application and configuration for their specific facility needs.

9 References

[1]Amazon, “AWS Documentation,” *Amazon.com*, 2019. https://docs.aws.amazon.com/ (accessed May 03, 2025).

[2]“Bruno - The Open Source API Client,” *Bruno Docs*, 2025. https://docs.usebruno.com/introduction/what-is-bruno (accessed May 03, 2025).

[3]“GoDaddy - Manage DNS records,” *Godaddy.com*, 2025. https://www.godaddy.com/help/manage-dns-records (accessed May 03, 2025).

[4]“MongoDB Documentation,” *Mongodb.com*, 2025. https://www.mongodb.com/docs/?msockid=348a20889edc617036a133089fd4600b (accessed May 03, 2025).

[5]MDN Contributors, “JavaScript,” *MDN Web Docs*, Sep. 25, 2023. https://developer.mozilla.org/en-US/docs/Web/javascript (accessed May 03, 2025).

[6]“Spring Framework Documentation :: Spring Framework,” *docs.spring.io*. https://docs.spring.io/spring-framework/reference/index.html (accessed May 03, 2025).

10 Appendices

Appendix 1 – Operation Manual

10.1.1 Create a new domain

1. If you are not Nathan Brockman, please contact him at mantisnb@iastate.edu in order to get your domain created correctly. (Keep in mind that your tagging system will NOT be able to be updated in the future, please ensure to provide the correct information when creating your domain.)
2. If you are Nathan Brockman, log in using your super admin credentials on the tracker.flutr.org/AdminLogin.html page.

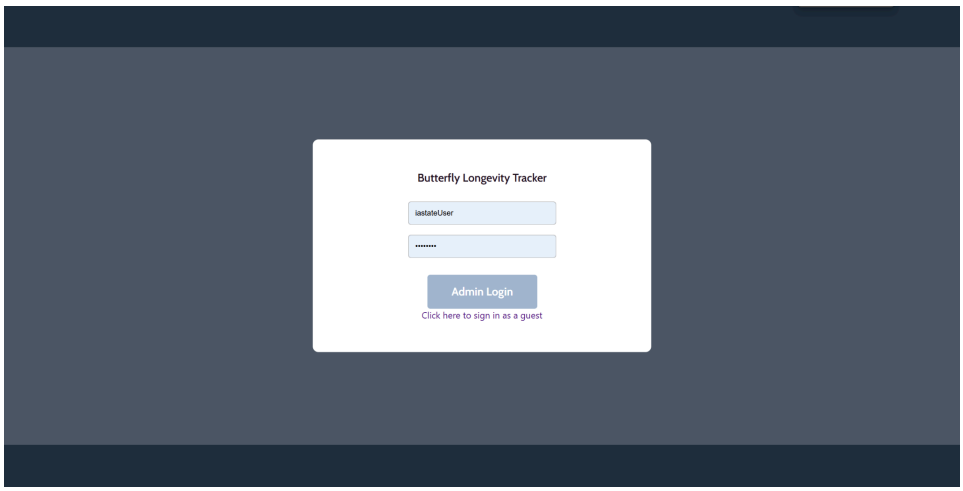


Figure 10.1

3. Navigate to the Create New Facility page by pressing the button located in the top right corner of the screen.

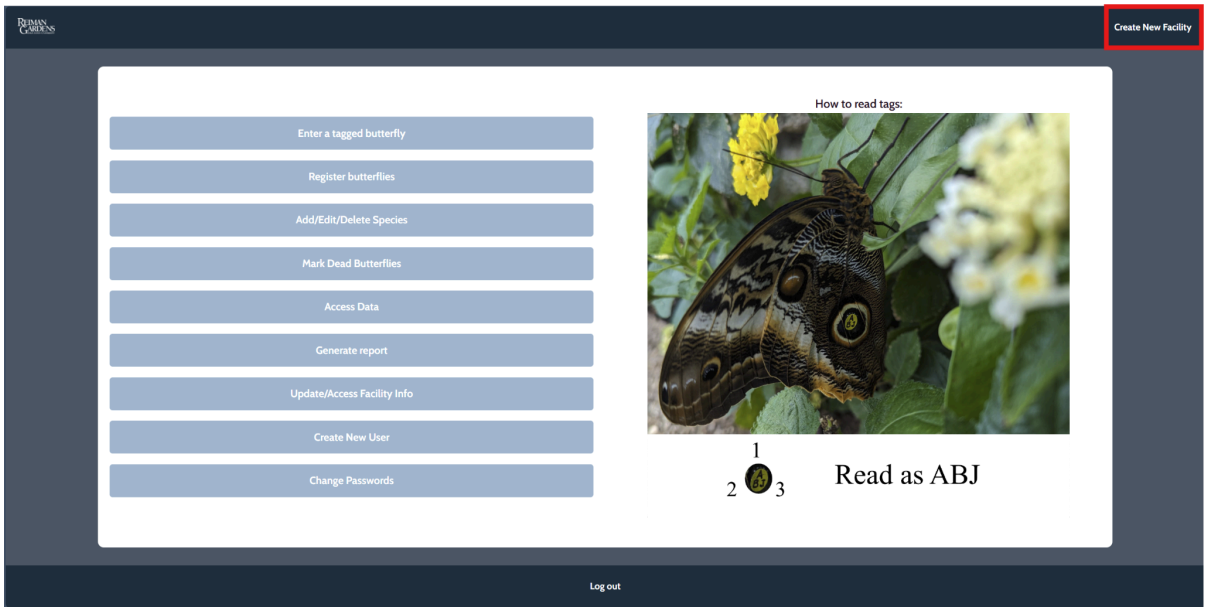


Figure 10.2

4. Select the necessary Tag Colors. e.g., if your domain uses sticker color as a unique identifier for your butterfly tags, then you will want to enable this option and select all of the colors your domain uses for tag color.

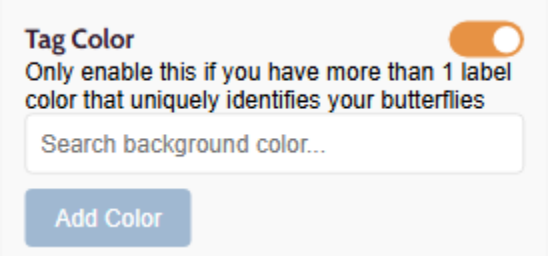


Figure 10.3

   a. Please do not enable this option if you only have one sticker color or do not use sticker color as a unique identifier.
   b. You must provide a minimum of two colors from the drop-down when the Tag Colors option is selected.

5. Select the necessary Tag Shapes. e.g., if your domain uses sticker shape as a unique identifier for your butterfly tags, then you will want to enable this option and enter all

sticker shapes that your domain uses to identify butterflies.

Figure 10.4

a. Please do not enable this option if you only have one sticker shape or do not use sticker shape as a unique identifier.
b. You must provide a minimum of two shapes when the Tag Shape option is selected.

6. Select the necessary Tag Foreground Colors. e.g., if your domain uses letter/symbol colors or a single colored dot in the foreground, then you will want to enable this option and enter all letter or dot colors that your domain uses to identify butterflies.

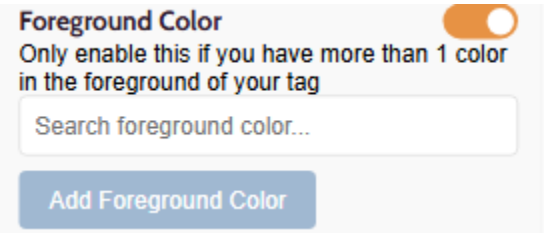


Figure 10.5

a. Please note that this option alone will only allow you to select ONE foreground color per butterfly. e.g., your tag might be "ABC Red", where A, B, and C are all in red font or have a red dot next to them.
b. Please do not enable this option if you only have one letter/dot color in your tags foreground, or do not use letter/dot color as a unique identifier.
c. You must provide a minimum of two colors when the Foreground Color option is selected.

7. Select the necessary characters to be allowed in your butterfly tags. e.g., if your domain uses uppercase letters and numbers to uniquely identify your butterflies, you will want

to toggle on both “Allow Uppercase Letters” and “Allow Numbers”

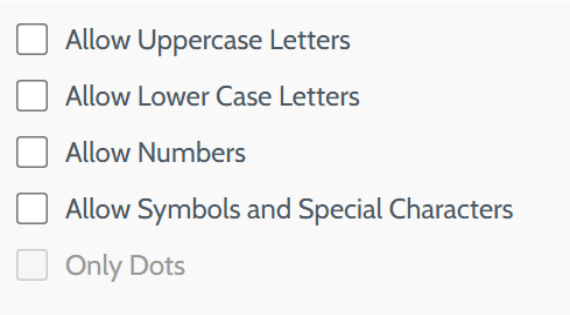


Figure 10.6

a. Besides “Only Dots”, these options can be used in any combination.
b. You are required to select at least one of these options
c. If your domain uses different colored dots in order to identify butterflies, enable foreground colors and select the “Only Dots” option.

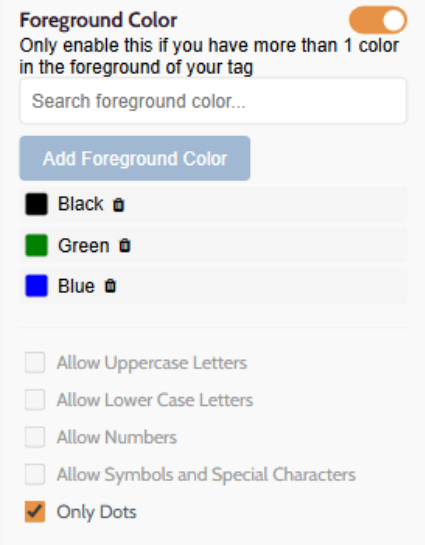


Figure 10.7

i. You must provide a minimum of two foreground colors
ii. Using “Only Dots” will not allow you to use any of the other character options.

8. Enter the number of characters or dots on each butterfly. e.g., if you use 4 unique characters or dots to identify your butterfly, you will enter 4.

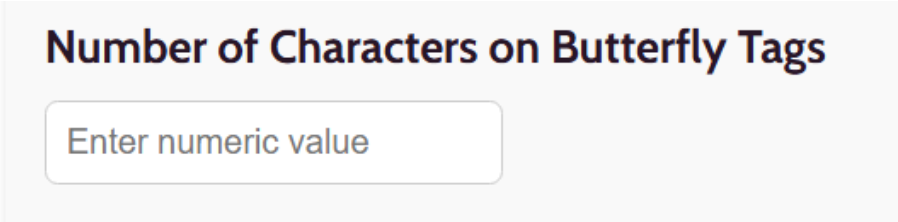


Figure 10.8

   a. Please note that this number will be a constant for all butterflies. You will not be able to have different length tag codes within a domain.
   b. This option is required in order to create your domain.

9. Enter information you would like your visitors to know about your facility in the About section.

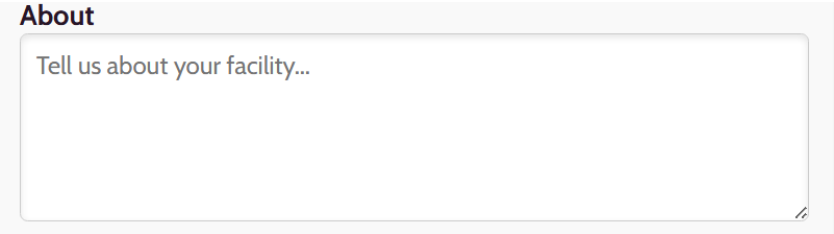


Figure 10.9

   a. This section is required and will be available to all users

10. Enter a username and password that you would like to use to log in as a dummy user for the domain.

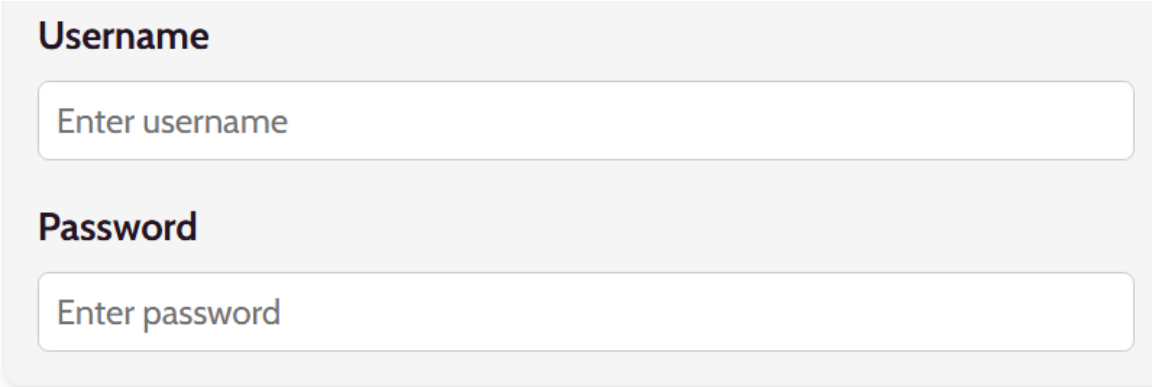


Figure 10.10

   a. This user will have domain admin permissions within your domain.
   b. You will log in to this account first and start creating any users you need in order to give each person an authenticated user account.

c. These fields are both required.

11. Upload the images corresponding to the tag instructions and logo to be displayed throughout the application.

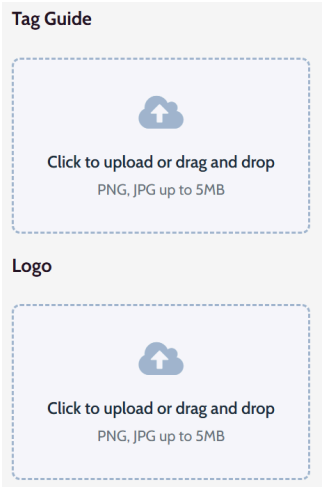


Figure 10.11

a. The tag guide image should be informative and explain how the user will read and enter butterfly tag information.
b. The logo should be associated with your butterfly exhibit, as it will be displayed throughout the web application.
c. Both images are required.

12. Choose a domain name and a domain ID unique to your butterfly exhibit

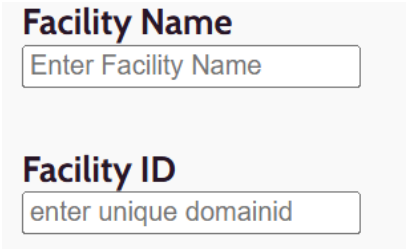


Figure 10.12

a. In the Facility Name option, you will enter the name of your exhibit. This will be displayed on the welcome page for guests.
b. In the Facility ID field, please enter some unique domain identification. This identifier must be all lowercase and contain only alpha characters.

13. Create your new domain and dummy user by pressing the submit button!

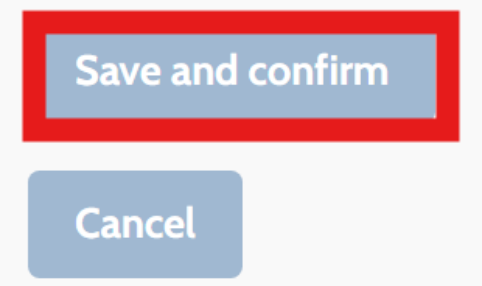


Figure 10.13

a. Confirmation messages will appear at the top of your screen if the creation was successful, otherwise, you will be displayed an error message along with what fields need to be filled.
b. The next step will be to log in to this dummy user, see section 10.1.2, Login as authenticated user.

10.1.2 Login as an authenticated user

1. Find the tracker.flutr.org/AdminLogin.html and enter your username and password credentials.

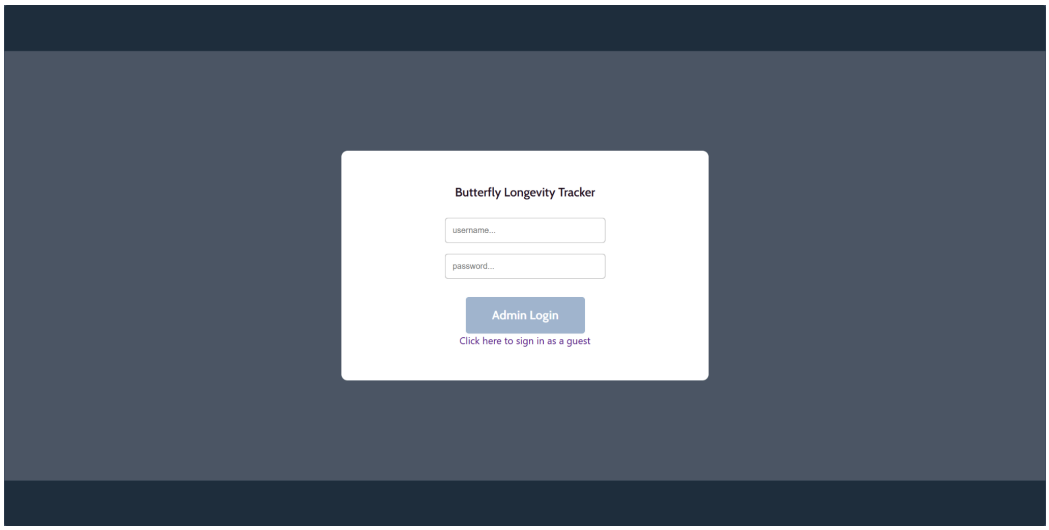


Figure 10.14

a. If you forget your password, please contact your domain administrator in order to get it changed.

10.1.3 Access domain information

1. Upon logging in to your authenticated admin account, navigate to the Update/Access Facility Info page.

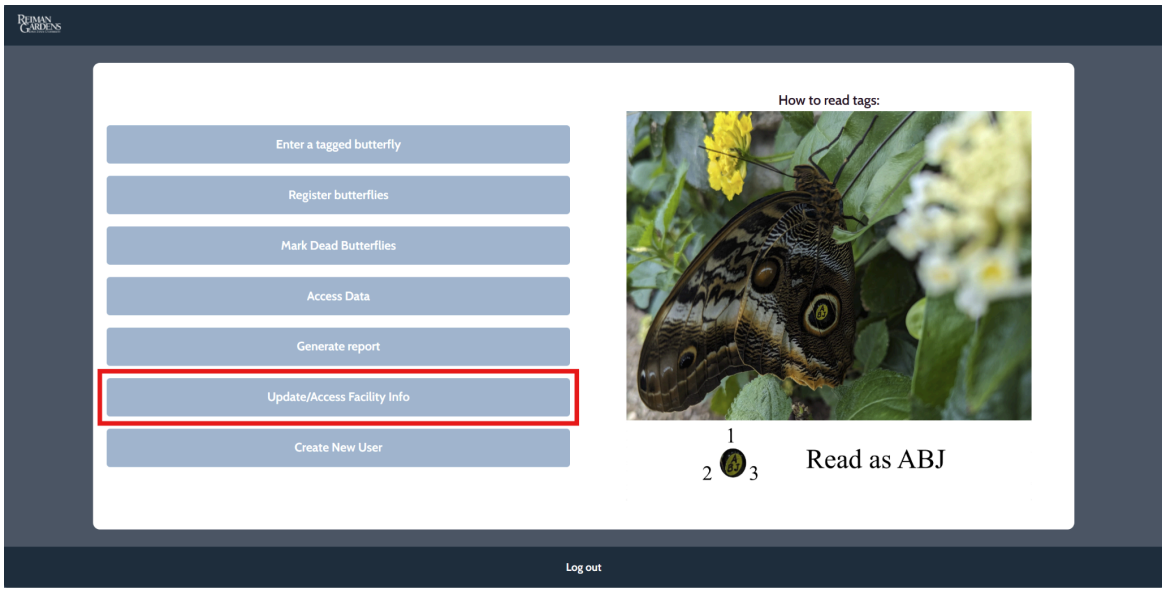


Figure 10.15

2. Here you will see all of the domain information, including a link to your domain's guest entry page as well as a generated QR code to that same link.

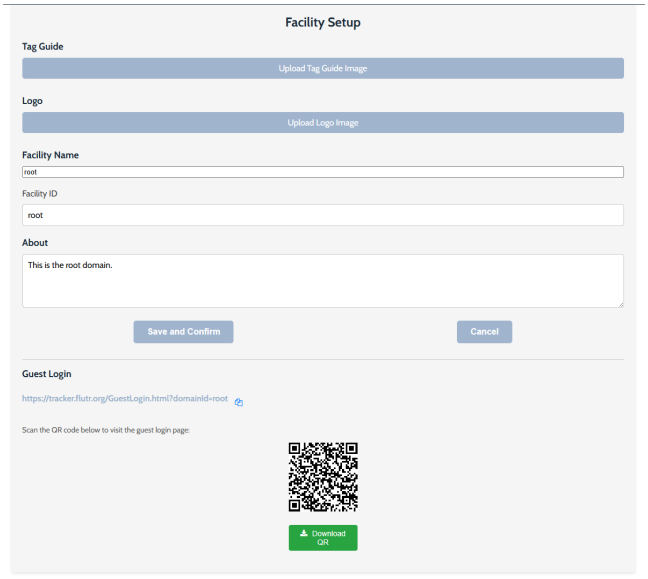


Figure 10.16

   a. You are able to download the QR code and print it out so that guests can scan it and have easy access to your domain's guest entry page.

10.1.4 Update domain information

1. Upon logging in to your authenticated admin account, navigate to the Update/Access Facility Info page.

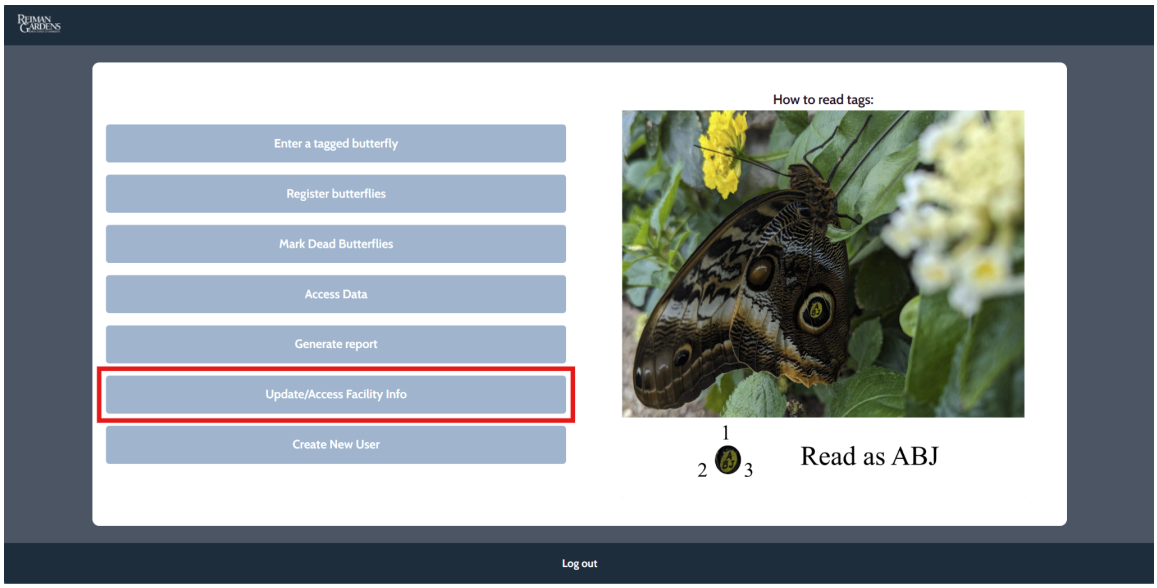


Figure 10.17

2. Edit any fields that you would like to update for your domain information. To confirm your changes, press the Save and Confirm button.

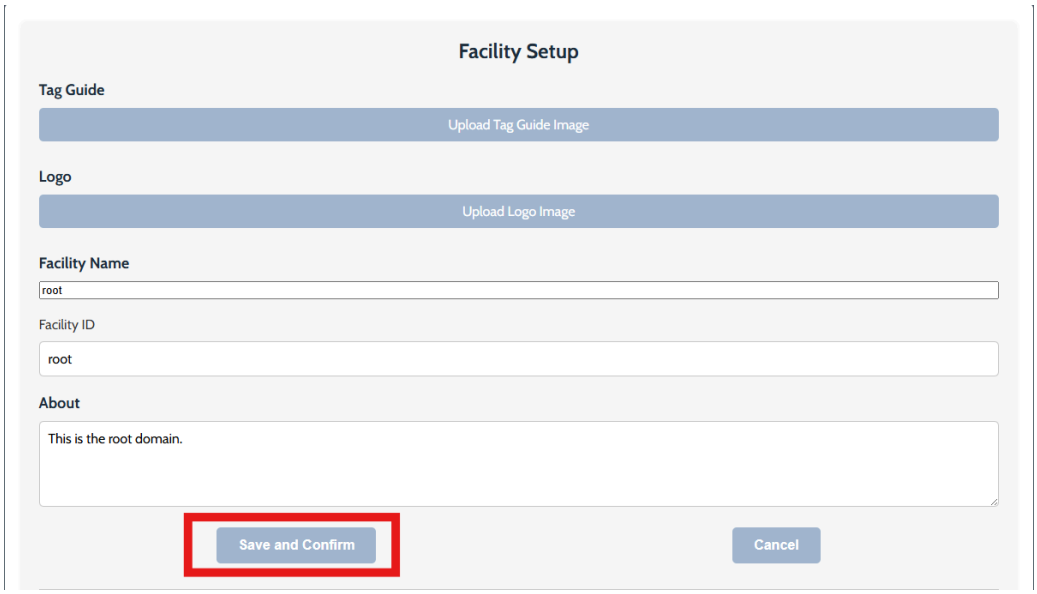


Figure 10.18

a. Please keep in mind that you are NOT able to change the Facility ID.
b. All changes made here will be global for all users within the domain.

3. Upon Successful completion, you will receive a confirmation message.

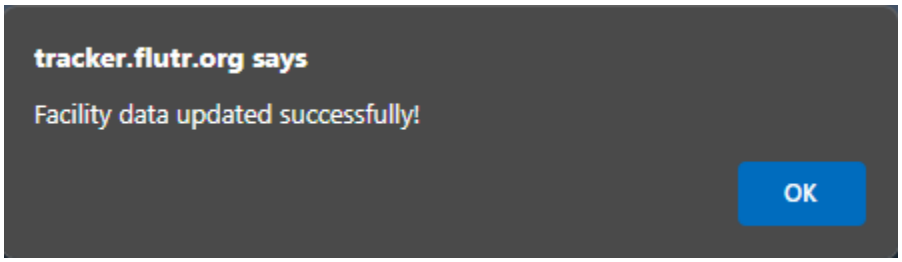


Figure 10.19

10.1.5 Create users within your domain

1. Upon logging in to your authenticated admin account, navigate to the Create New User page.

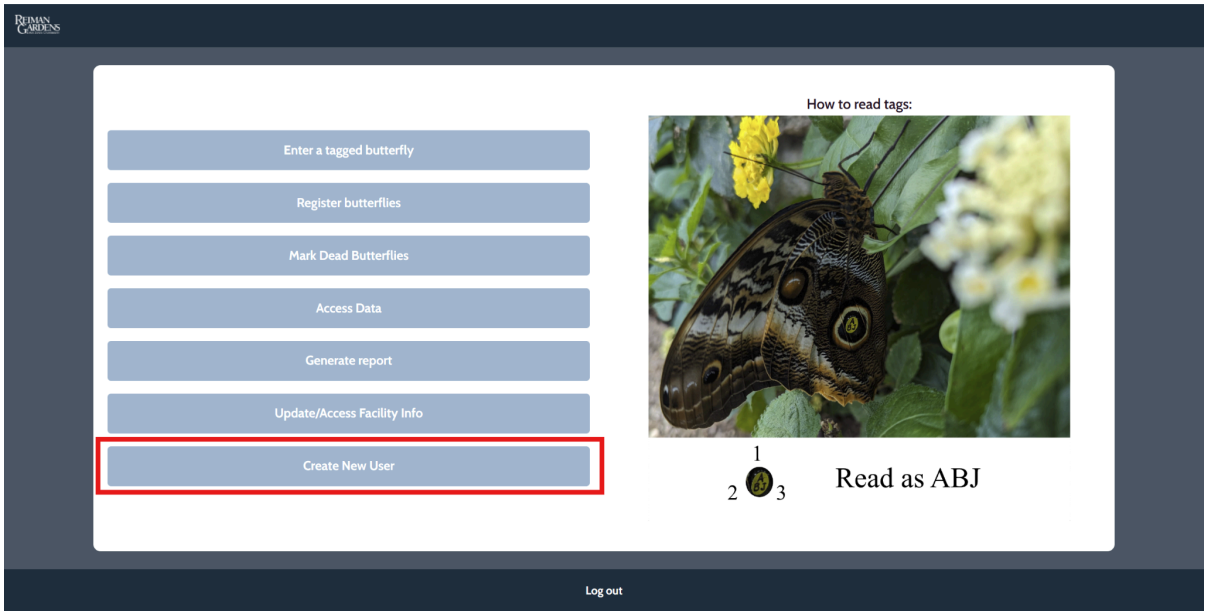


Figure 10.20

2. Fill out the Create User form in order to make a new authenticated user for your domain, and press submit when you are finished.

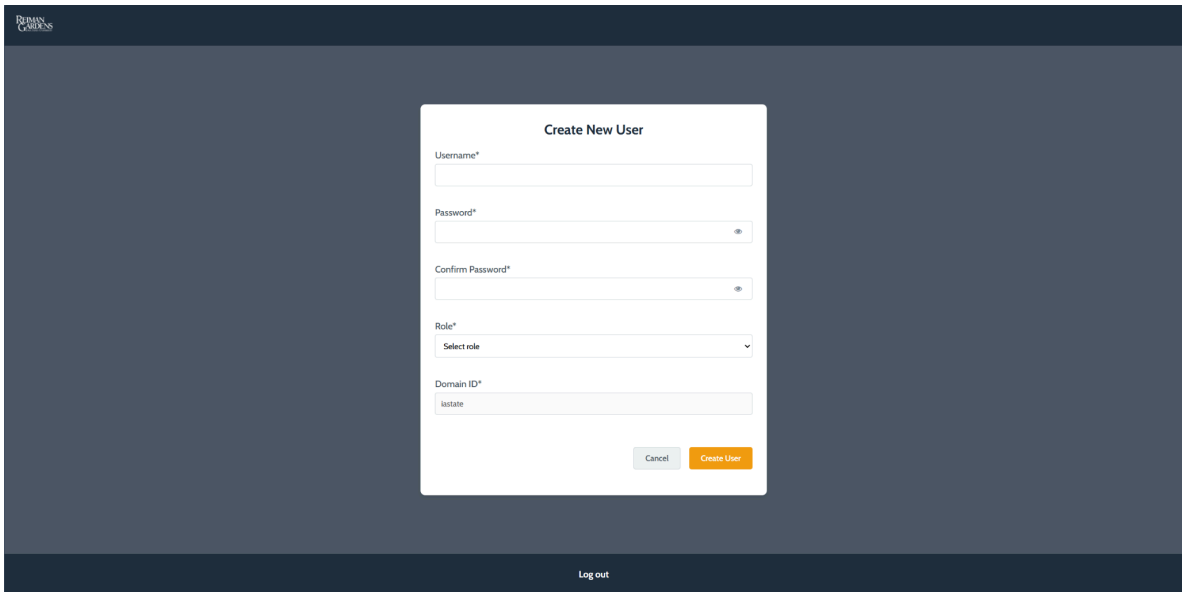


Figure 10.21

   a. Keep in mind that all fields are required and final; you will not be able to update the user's role later.

10.1.6 Create users within other domains (Super Admin users only)

1. Upon logging in to your authenticated super admin account, navigate to the Create New User page.

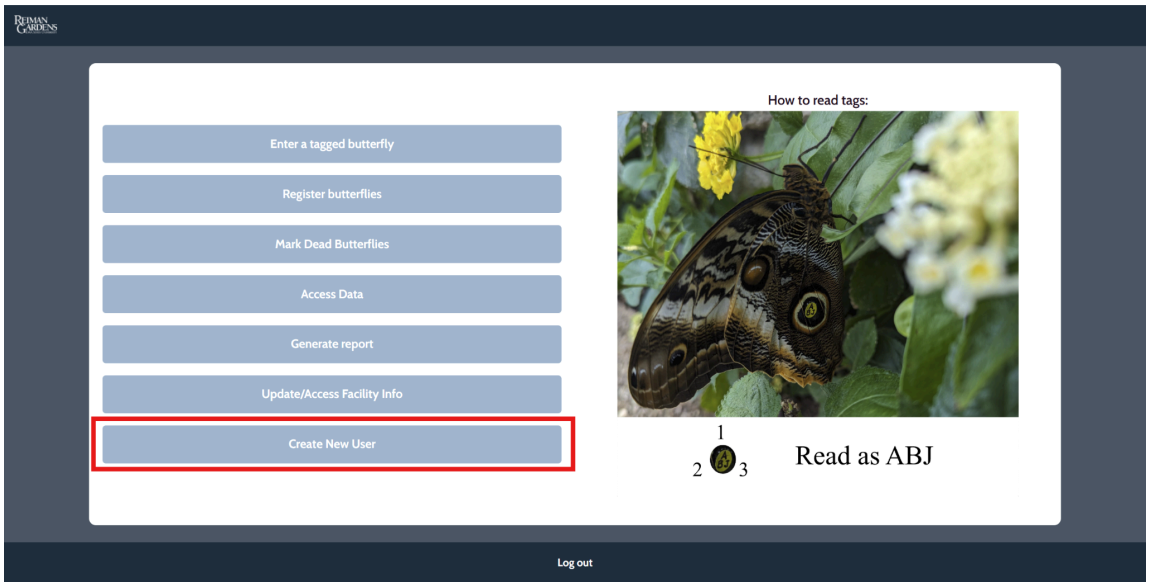


Figure 10.22

2. You have now gained access to create Super Admin user accounts, which can be found under the role selection.

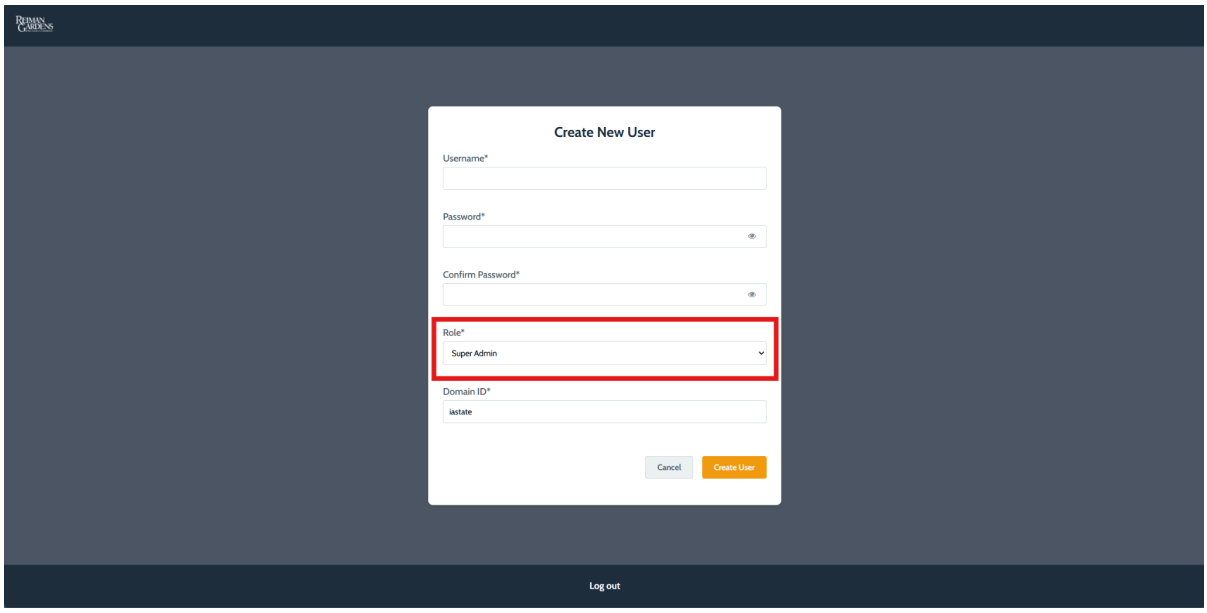


Figure 10.23

   a. Please only create Super Admin users for trusted individuals, and note that they will have the same powers as you.

3. You will now also be able to edit the Domain ID field, and you will be able to specify which domain you would like this user to be under.

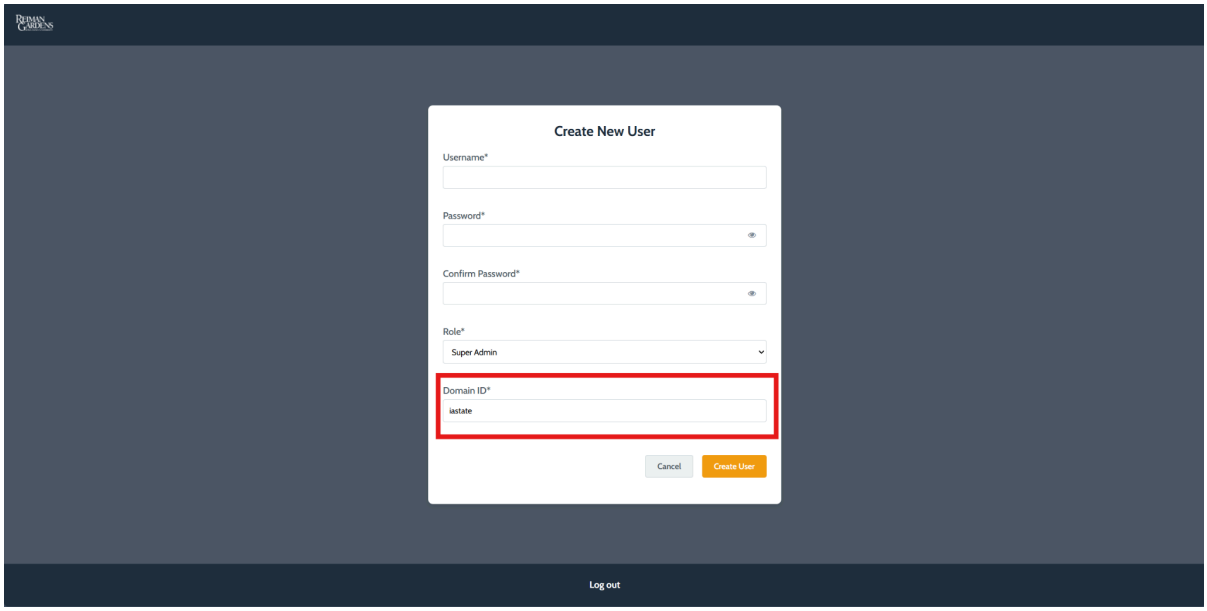


Figure 10.24

a. Please use your powers responsibly, and we ask that you confirm with the domain administrator that you are creating an account prior to submitting new user accounts.

10.1.7 Change user passwords (Super Admin users only)

1. Upon logging in to your authenticated super admin account, navigate to the Create New User page.

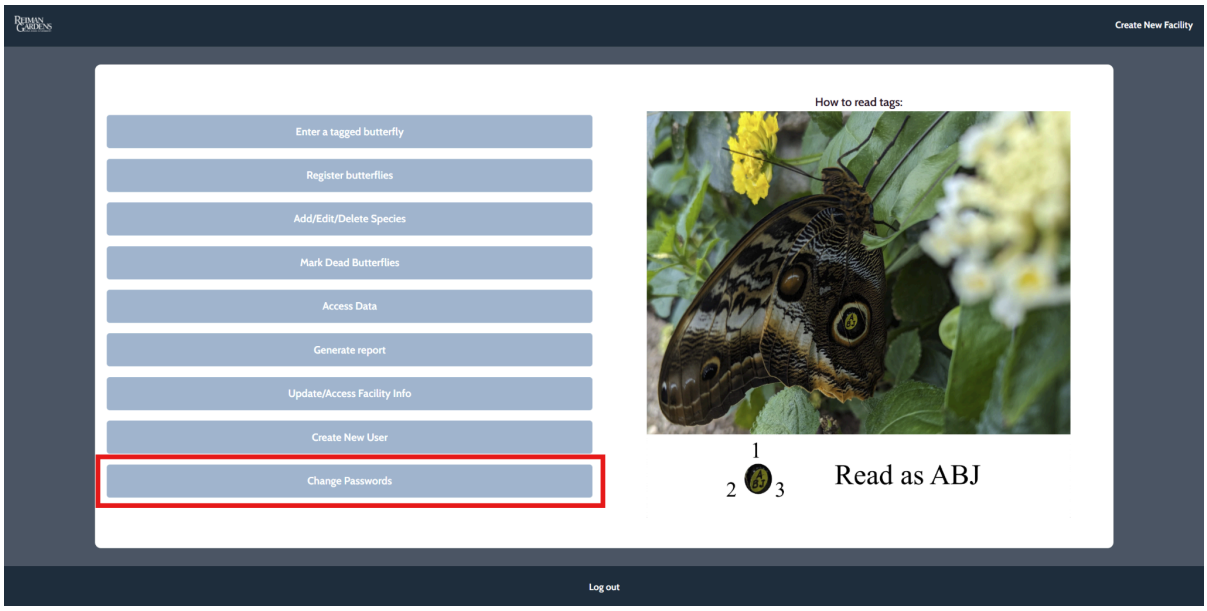


Figure 10.25

2. Enter the Username who needs to have their password changed. Enter the new password for the user and click on the Update Password button to confirm the

changes.

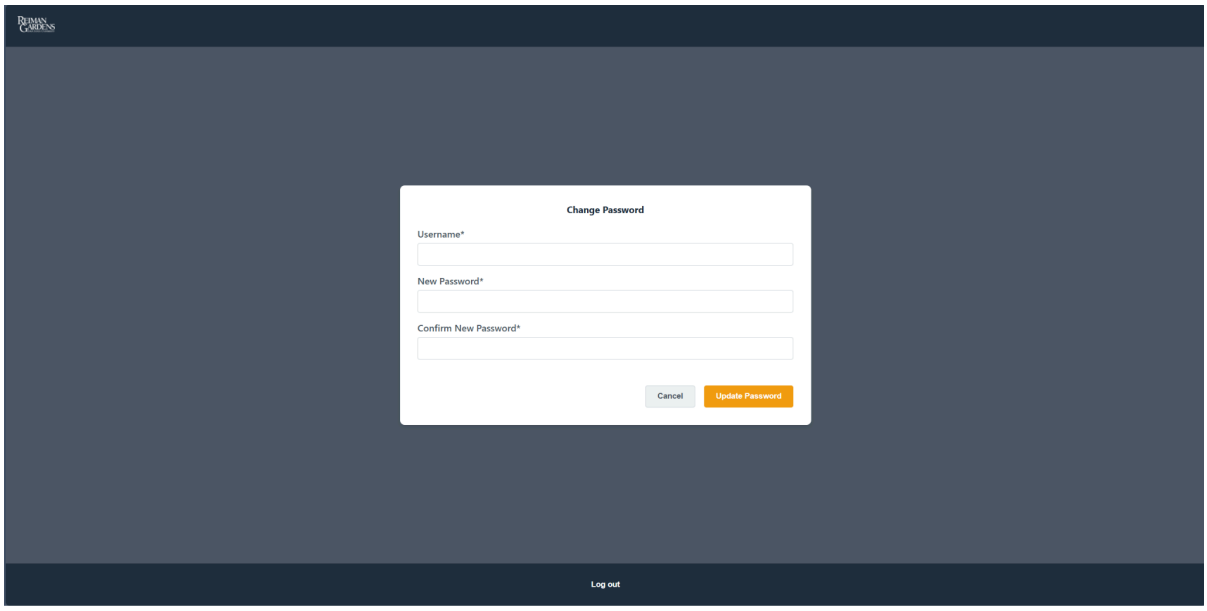


Figure 10.26

a. Note that this password update will be effective immediately.

10.1.8 Registering Butterflies within your domain

1. Upon logging in to your authenticated admin account, navigate to the Register Butterflies page.

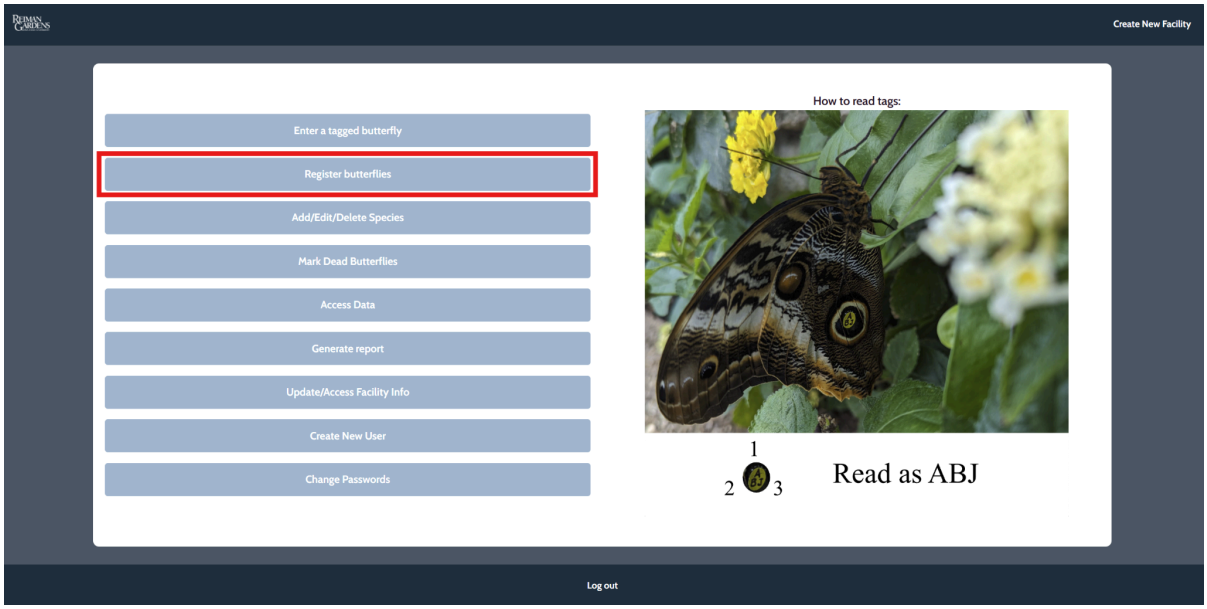


Figure 10.27

2. The first thing to do will be selecting the date and time at which you want to register the date to.

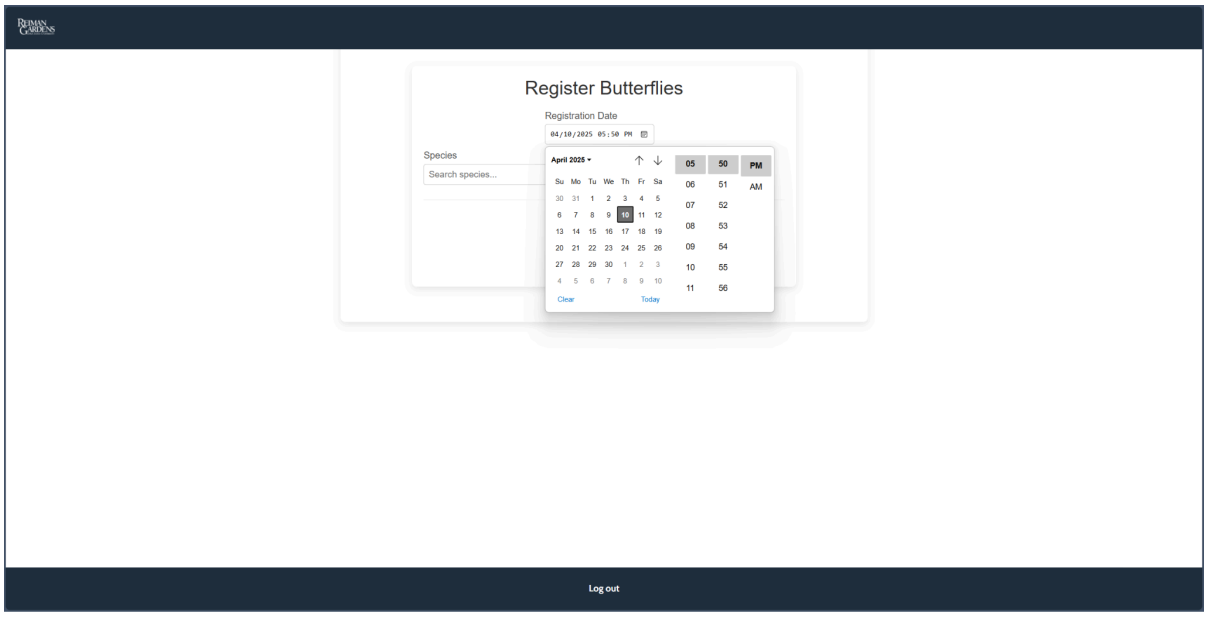


Figure 10.28

   a. You will be able to choose previous dates if your system is down in order to track the proper butterfly life span time.

3. Enter the species name and Tag Code definition corresponding to the butterfly that you want to register.

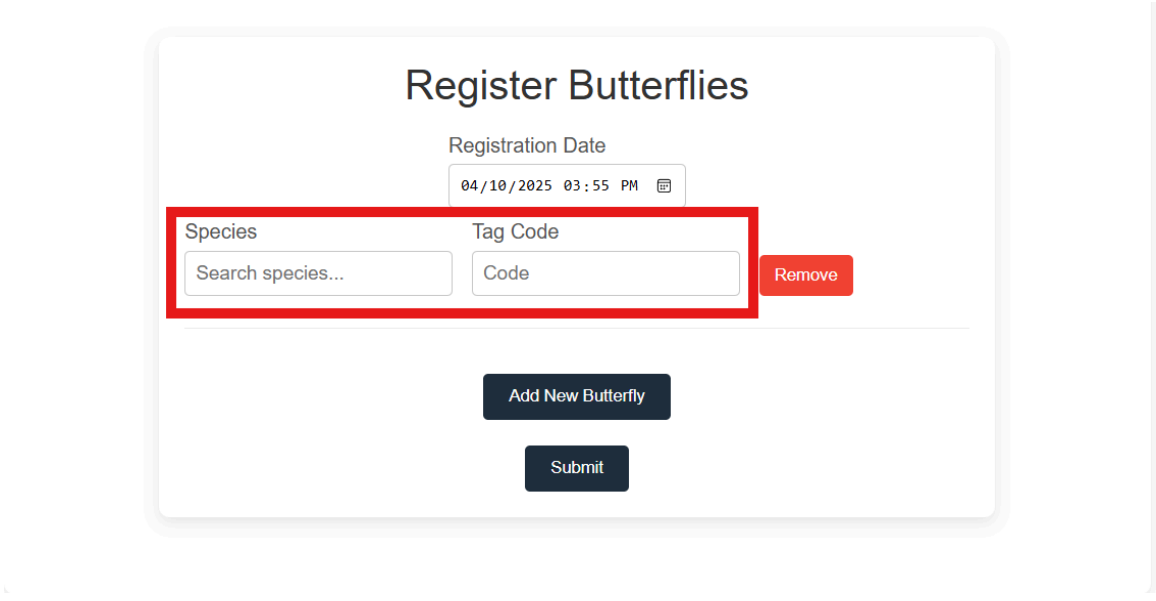


Figure 10.29

    a. You will only be allowed to enter tags that correspond to the tag definition specified while setting up your domain.
4. To register more than one butterfly at a time, press the “Add New Butterfly” button, and another butterfly registration row will be added. You can easily remove these rows by pressing the red “Remove” button to the right of the corresponding row you want to remove.

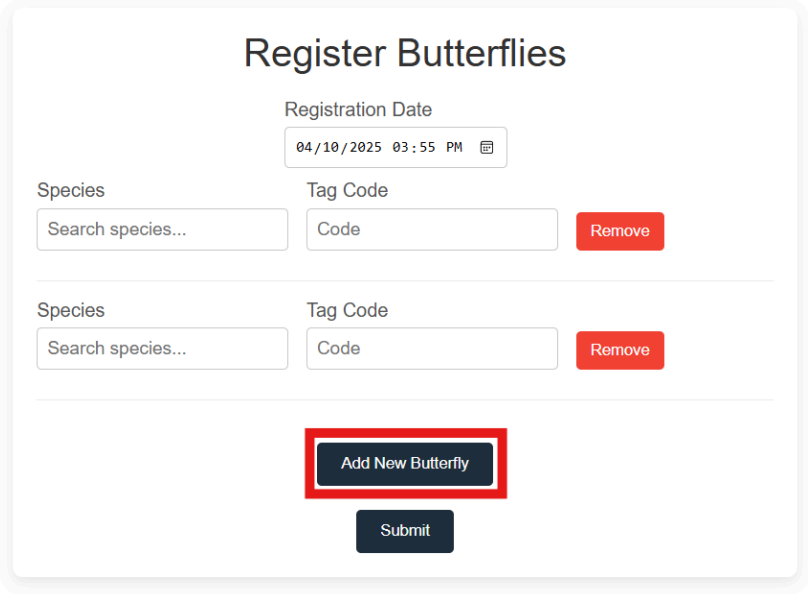


Figure 10.30

    a. Note that you can not have any blank rows in your form when you press submit.
    b. The date and time variable will be the same for each butterfly when they are submitted together. It is auto-populated to the current time in your time zone.
5. Press the “Submit” Button to add these new butterflies to the database. Everyone will now be able to enter their codes in the sighting entry page.

10.1.9 Marking butterflies as dead

1. Upon logging in to your authenticated admin account, navigate to the Mark Dead Butterflies page.

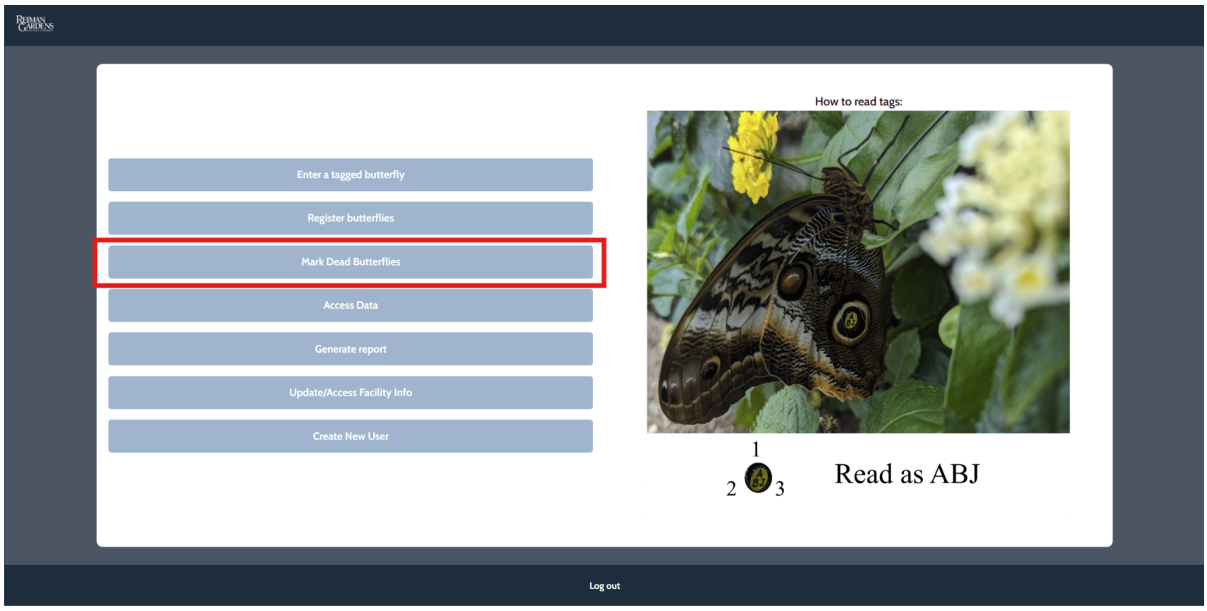


Figure 10.31

2. Enter the Tag Code of the butterfly that you would like to mark as dead.

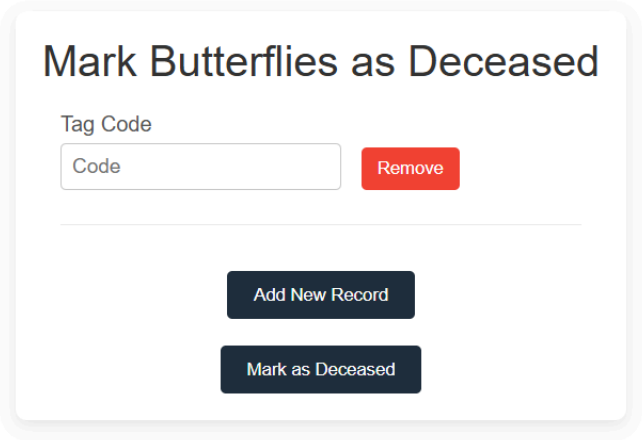


Figure 10.32

3. If you would like to mark multiple butterflies as dead, click the "Add New Record" button in order to add another row and enter another tag. Press the red remove button

in the corresponding row to remove a butterfly tag from the mark as dead form.

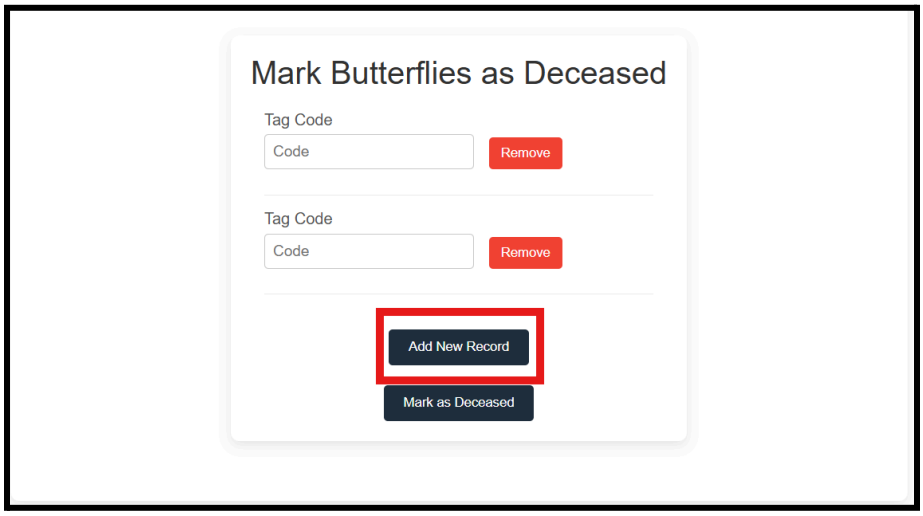


Figure 10.33

4. Press the "Mark as Deceased" button to confirm the death of these butterflies in the database.

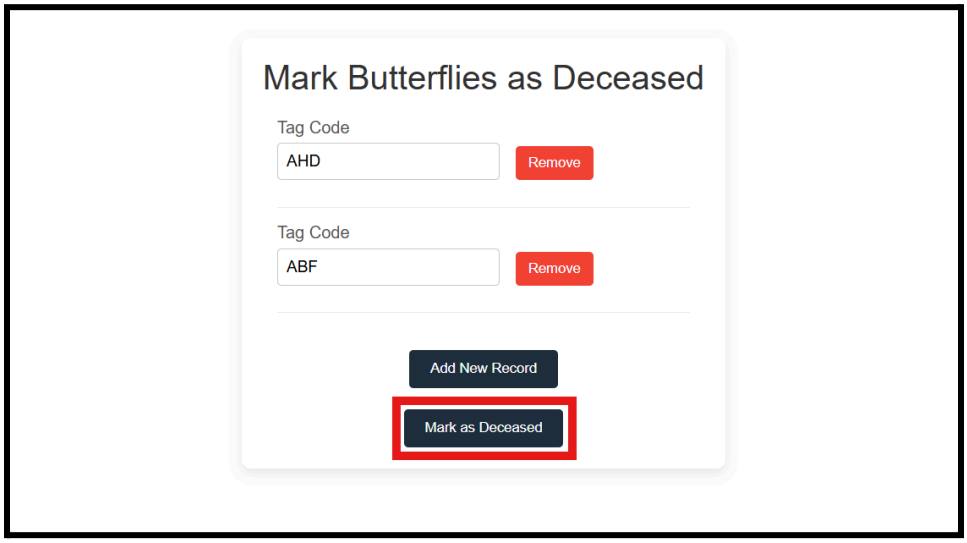


Figure 10.34

   a. The death date and time will be set to the local time of the user when the "Mark as Deceased" button is pressed.

10.1.10 Adding new species of butterflies (Super Admin users only)

1. After logging in, select the 'Add/Edit/Delete Species' button from the Admin Home page to be redirected to the Edit Butterflies page.

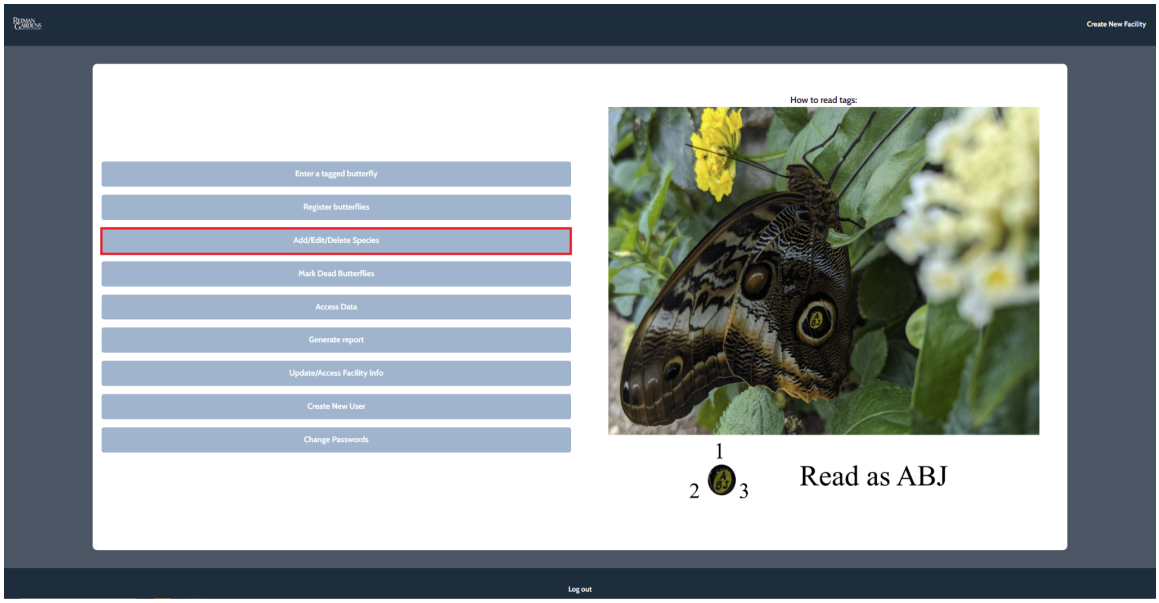


Figure 10.35

2. On the next page, select the 'Add' tab at the top of the window (1). Next, fill in the information for the new butterfly species you want to add, which includes the 'Common Name' and 'Species Name' fields (2).

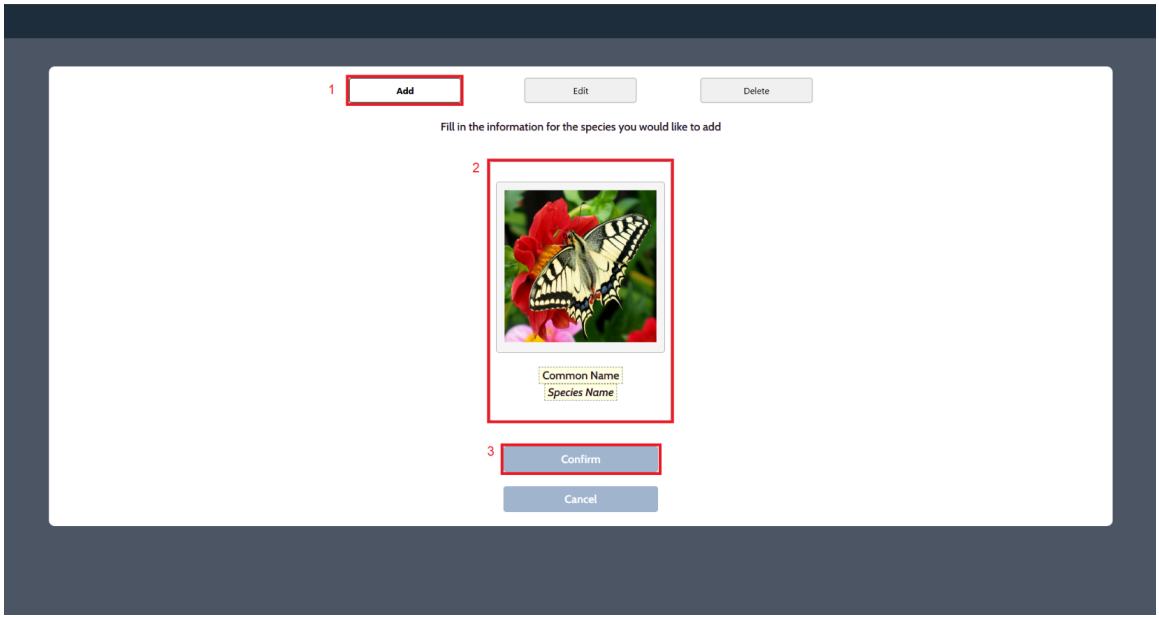


Figure 10.36

3. Once you have filled in the specified butterfly information, click the 'Confirm' button (3) to submit the new species to the database. This new species will now be fully functional.

Note: This application is using a previous senior design team's Digital Ocean image bucket to provide unique species images to users. If you desire to add a unique species image, you must add a picture to that Digital Ocean image bucket in the following format: <Species Name>_closed.JPG

For example, the Acrea anemosa species has the following image within the digital ocean bucket: Acrea anemosa_closed.JPG

If images are uploaded in this format, the app will automatically use the new images for the unique species; if not, the app will use a default image of a butterfly whenever possible.

10.1.11 Editing existing species

1. After logging in, select the 'Add/Edit/Delete Species' button from the Admin Home page to be redirected to the Edit Butterflies page.

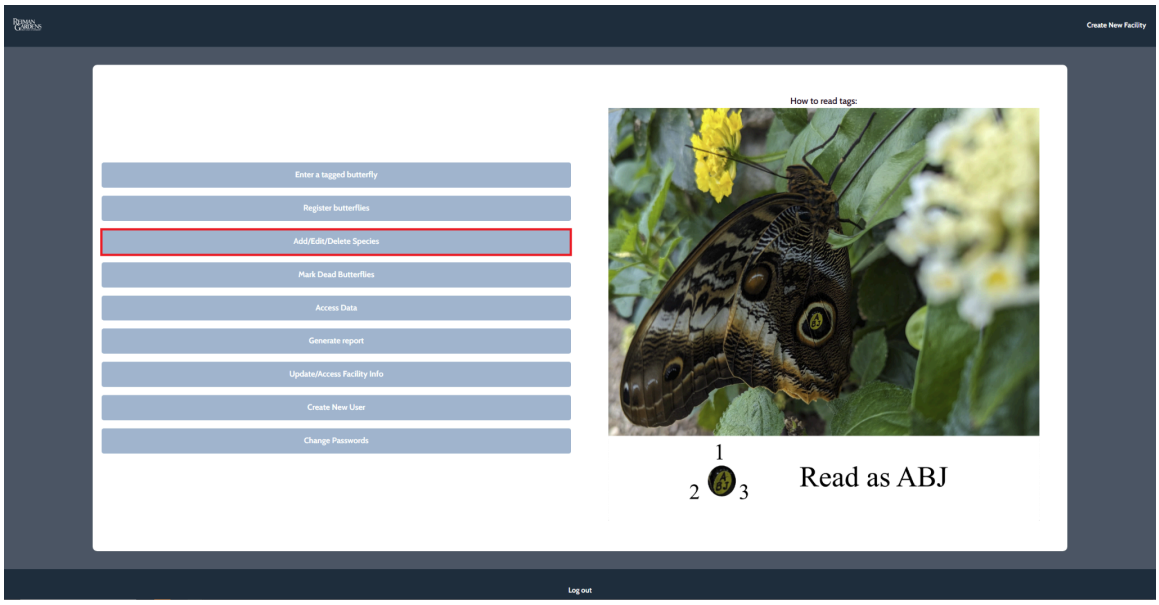


Figure 10.37

2. Ensure you have selected the 'Edit' tab (1) at the top of the screen. Next, in the search field (2), enter the species name that you would like to update. When you select a species from the search drop-down, the page will auto-populate with the species information (3).

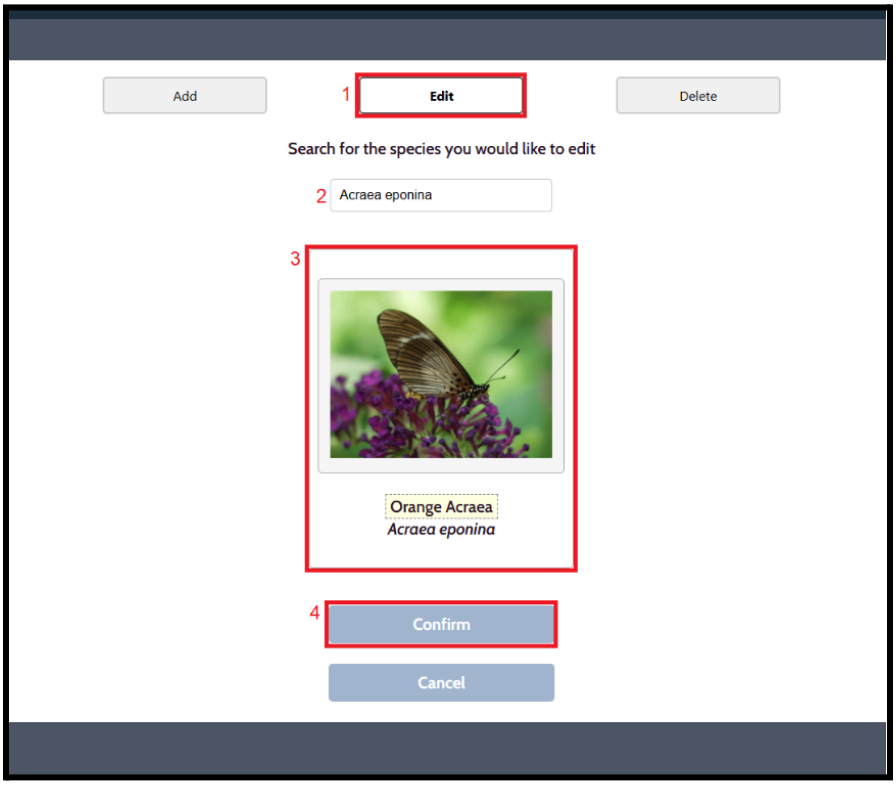


Figure 10.38

3. Within the species information section (3), each field will now be editable. Here you can change the information with the desired new information for the butterfly species. When you have edited the desired fields, press the 'Confirm' button (4) to submit the information changes to the database.

   Note: If you wish to change the species name, it is recommended to delete the species and recreate it from the 'Add Species' tab.

10.1.12 Deleting species of butterflies

1. After logging in, select the 'Add/Edit/Delete Species' button from the Admin Home page to be redirected to the Edit Butterflies page.

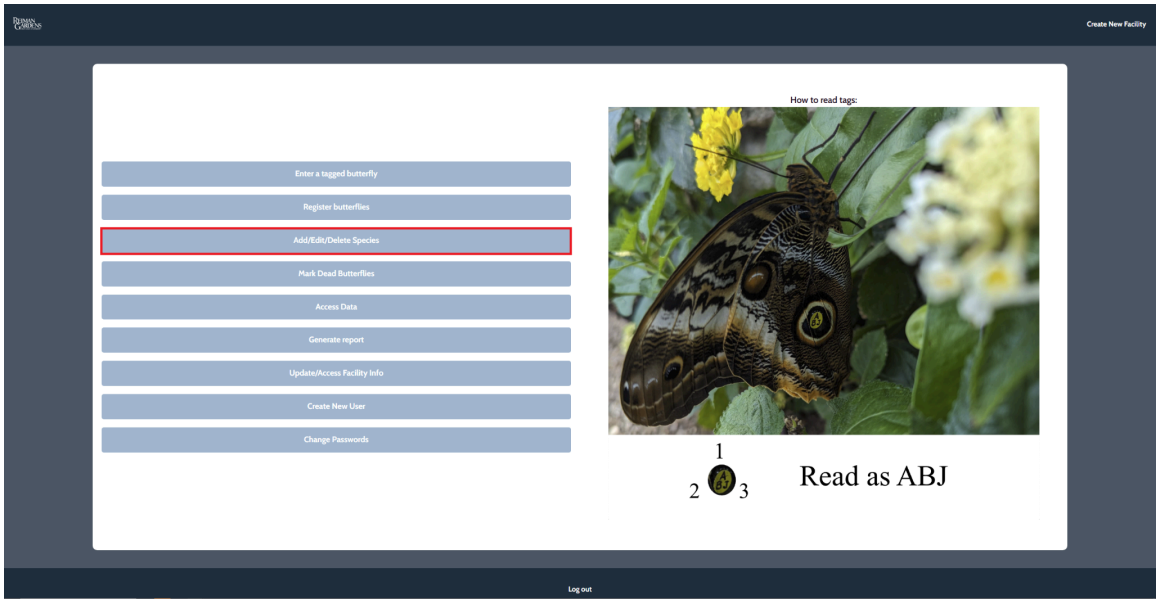


Figure 10.39

2. Ensure you have the 'Delete' tab (1) selected at the top of the screen. Next, in the search field (2), enter the species name that you would like to delete. When you select a species from the search drop-down, the page will auto-populate with the species information (3).

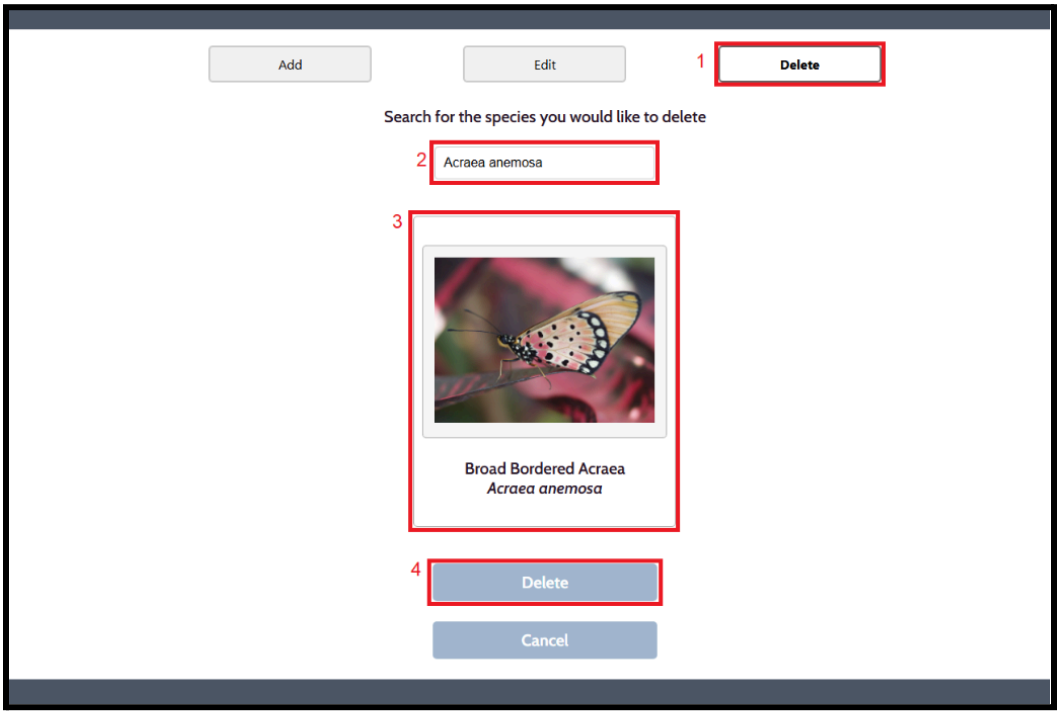


Figure 10.40

3. Ensure the populated butterfly information (3) contains the species that you want to delete. When you have confirmed that the species is the correct species that you are intending to delete, hit the 'Delete' button (4) to remove the species from the database.

10.1.13 Sighting Butterflies as an authenticated user

1. After logging in, press the 'Enter a tagged butterfly' button to navigate to the next page.

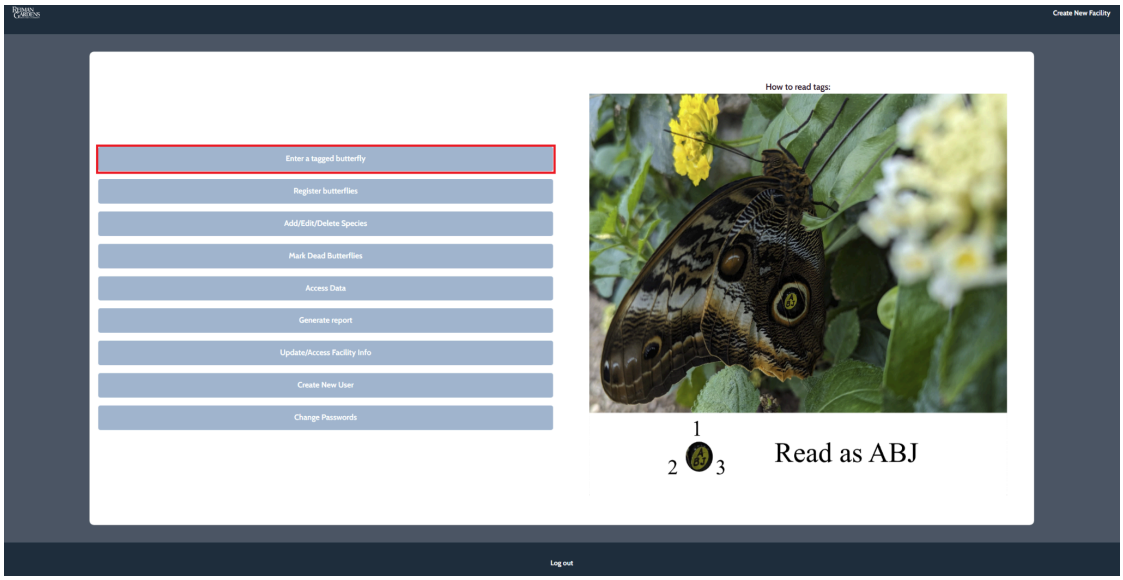


Figure 10.41

2. Next, enter the desired tag into the entry box.

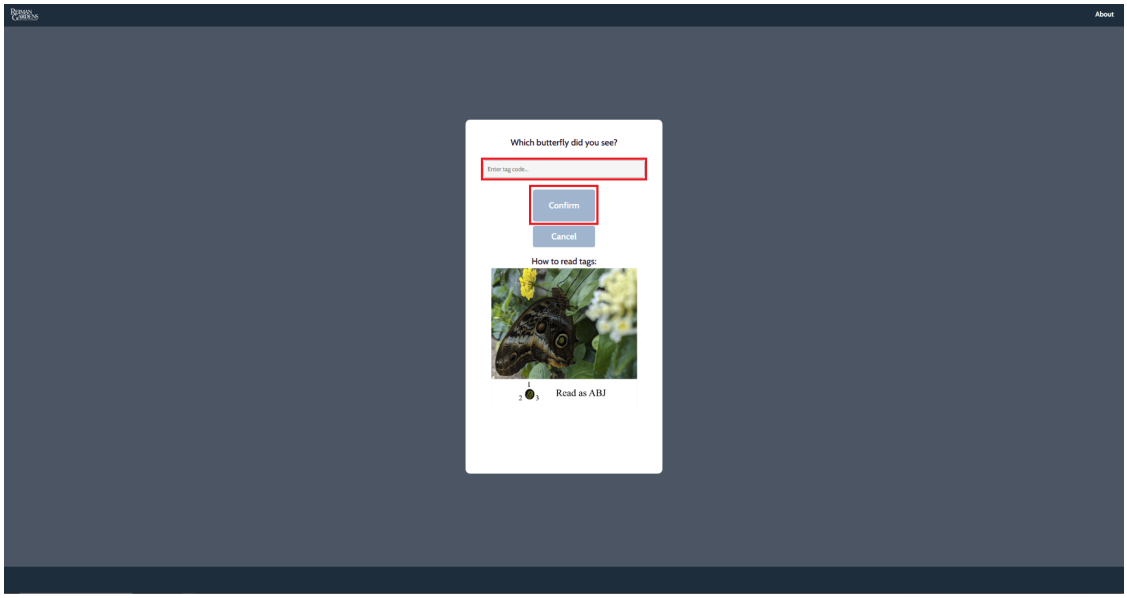


Figure 10.42

3. After entering the desired tag, press the confirm button to enter the sighting. Upon confirmation, you will be redirected to the next screen with the butterfly information displayed. Note that if you enter a tag that does not exist, you will be prompted with an error message.

10.1.14 Access sighting data within your domain

1. After logging in, press the 'Access Data' button on the Admin Home page to be redirected to the database view page.

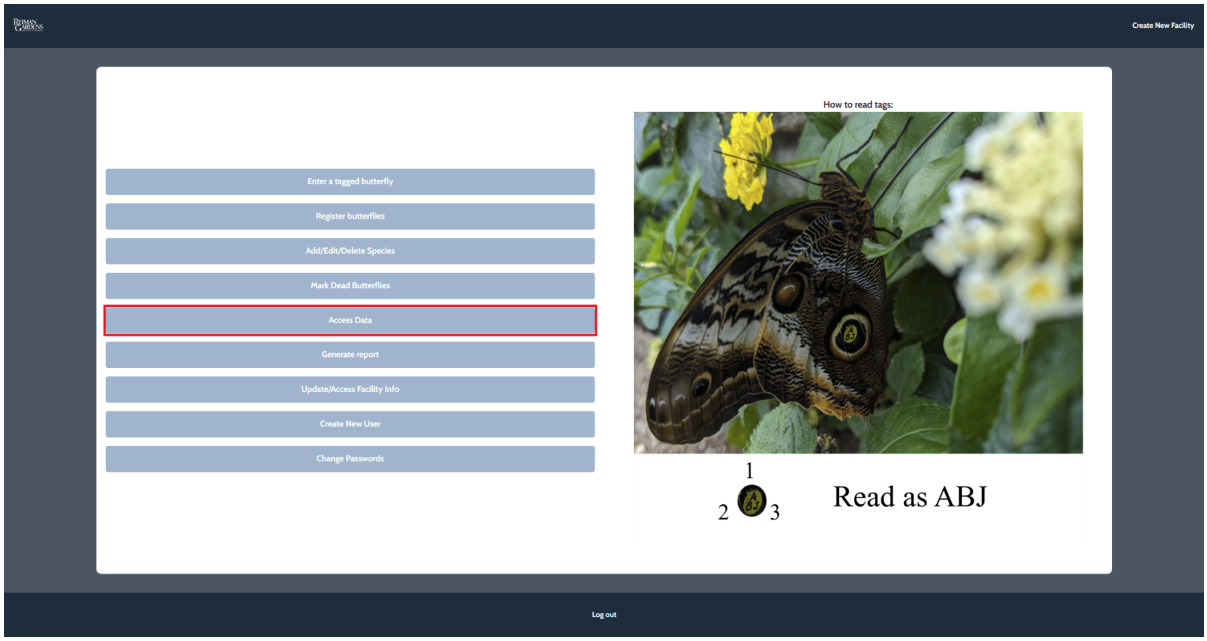


Figure 10.43

2. After you have been redirected, you will be presented with all the sighting entries from the domain your account is associated with. From here, you can click on the columns to sort by what you prefer, or you can search by any field that the top of the screen.

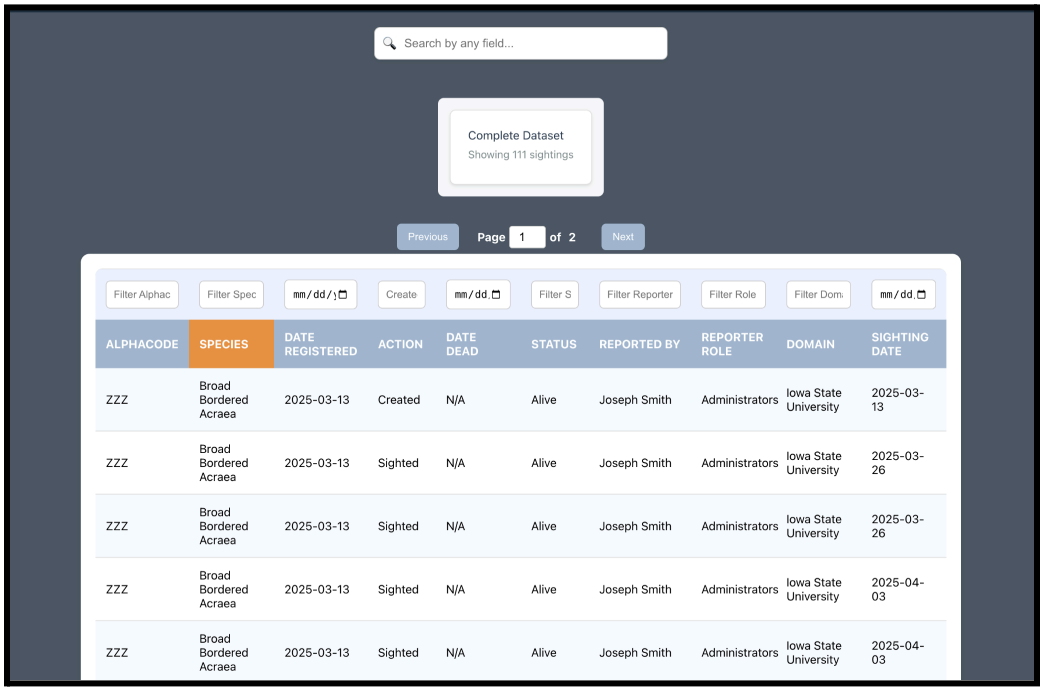


Figure 10.44

Note: If you are logged into a Super Admin account, you will automatically be returned data from **all** domains (depending on the number of sightings total, can take up to 20 seconds). If you want to see individual domains, use a non-super admin account, or use the Generate Report features below.

10.1.15 Generate a report on your domain data

1. After logging in, click on the 'Generate report' button to be navigated to the next page.

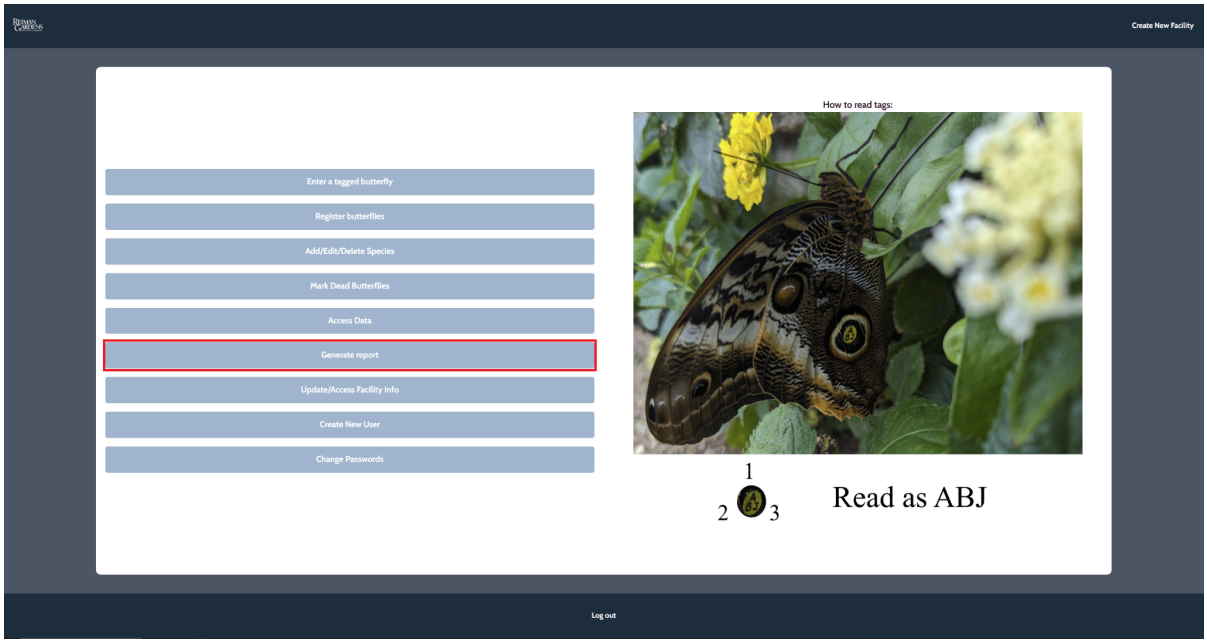


Figure 10.45

2. When generating a report, first select a report type from the drop-down menu. Then enter the desired information in the lower section (changes based on report type)

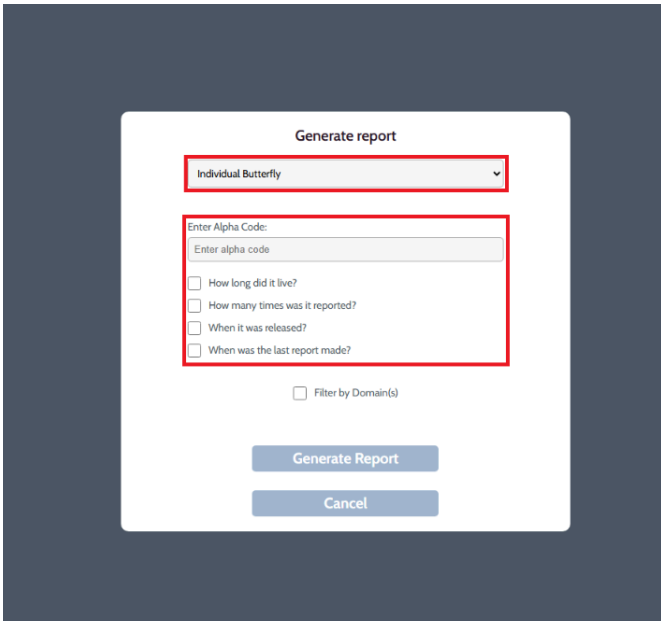


Figure 10.46

3. When you have selected the report type you want and entered the desired information in the lower section, click the generate report button to be taken to the report. The

selected information will be displayed at the top of the report along with all the sightings that meet the criteria of the report.

Figure 10.47

10.1.16 Generate report on other domains' data (Super Admin users only)

1. After logging in, click on the 'Generate report' button to be navigated to the next page.

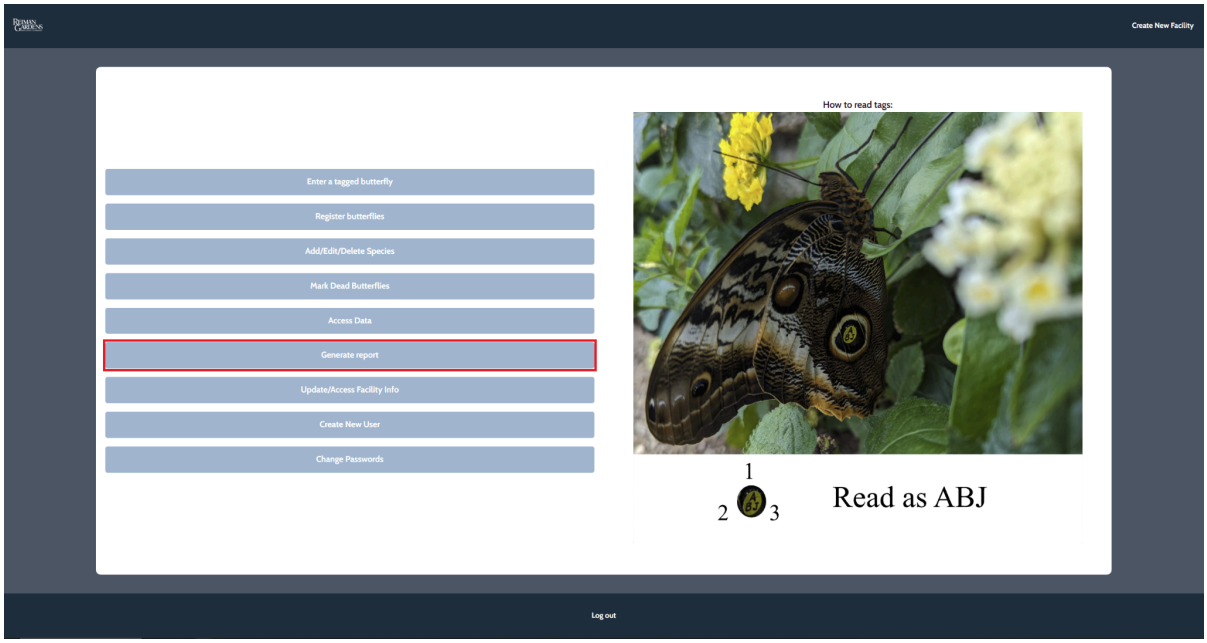


Figure 10.48

2. When generating a report, first select a report type from the drop-down menu. Then enter the desired information in the lower section (changes based on report type).
3. To filter by domain, select the domain filter at the bottom. You will then see a search bar where you can select any number of domains to include in your search. If you do not select any domains to filter as a Super Admin, you will automatically receive information only from your domain.

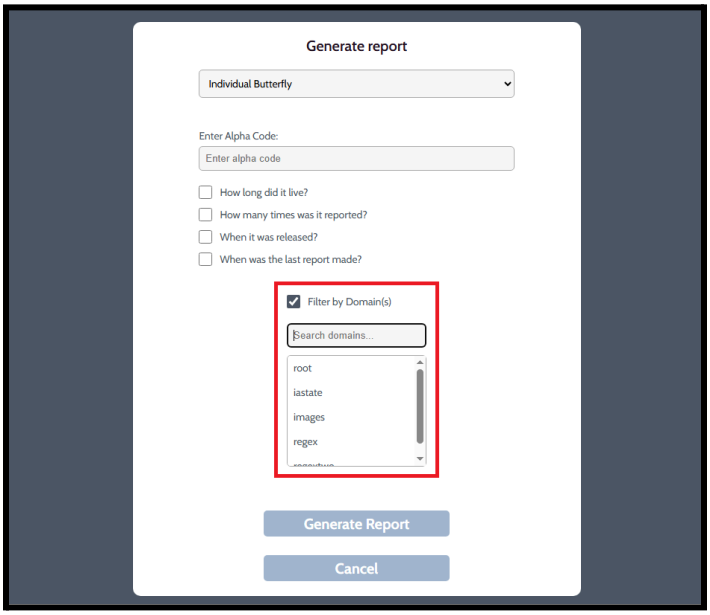


Figure 10.49

10.1.17 Logout

1. Upon logging in as an authenticated admin or super admin, you will see the logout button at the bottom of your screen. Press it to log out and return to the Admin Login

page.

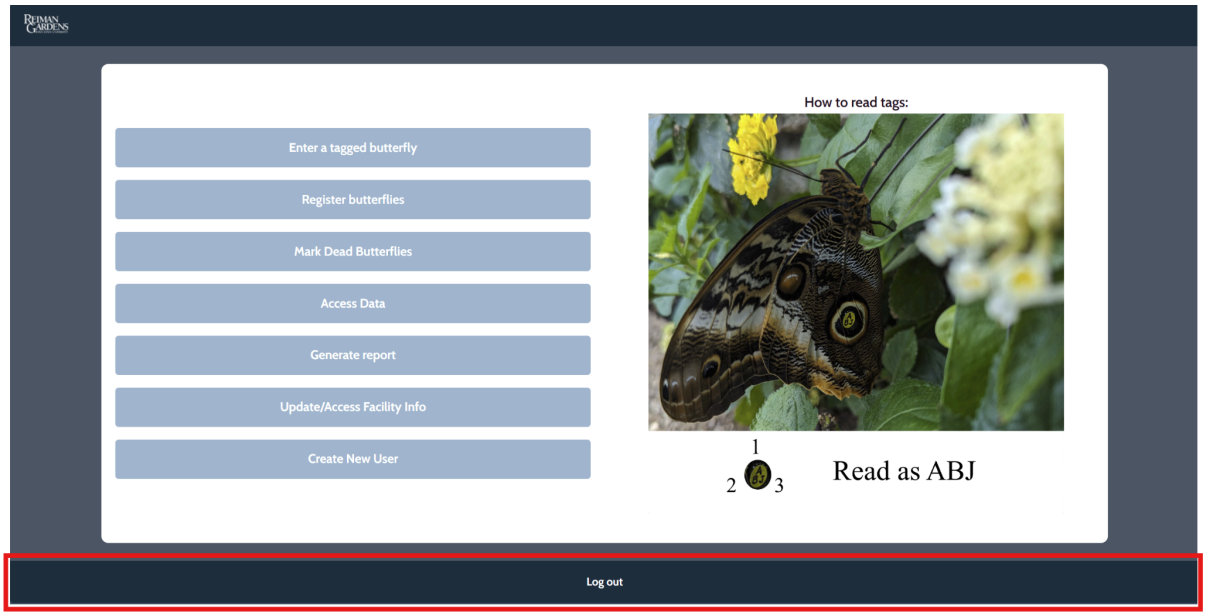


Figure 10.50

10.1.18 Enter as a guest

1. When visiting the butterfly exhibit, find the QR code provided by the butterfly exhibit and scan it.

Figure 10.51

2. Enter your name and press the “Enter as Guest” button in order to enter the domains page as a guest.

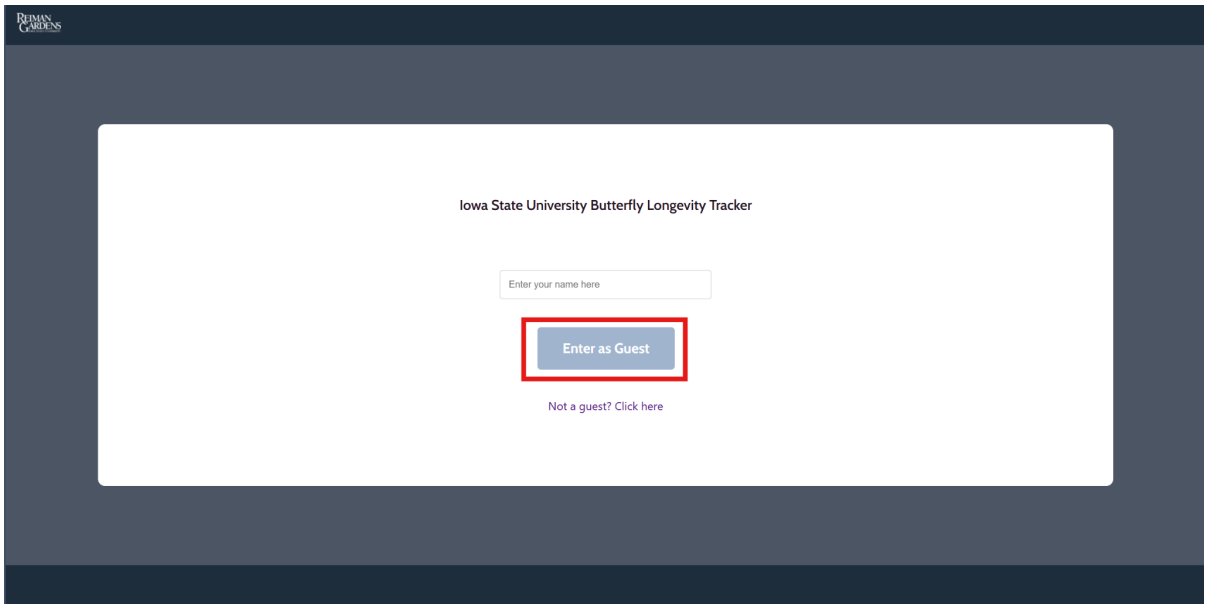


Figure 10.52

10.1.19 Tag a butterfly

1. If you are a guest, follow the instructions on 10.1.18, Enter as a guest. If you are a docent, log in with your authenticated username and password. You will be auto redirected to the same landing page as the guest user. If you are a domain or super admin, log in with your credentials and press the “Enter a tagged butterfly” button.

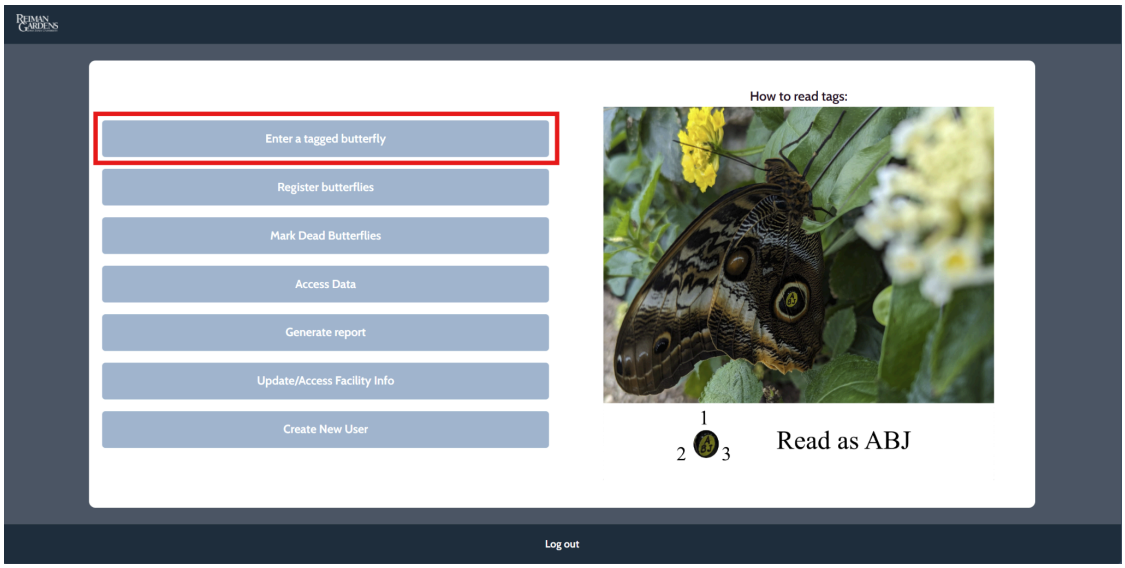


Figure 10.53

2. You will be met with the ability to enter the tag code of a butterfly you spot, along with the tagging instructions provided by the domain. Click the “Confirm” button to submit your sighting.

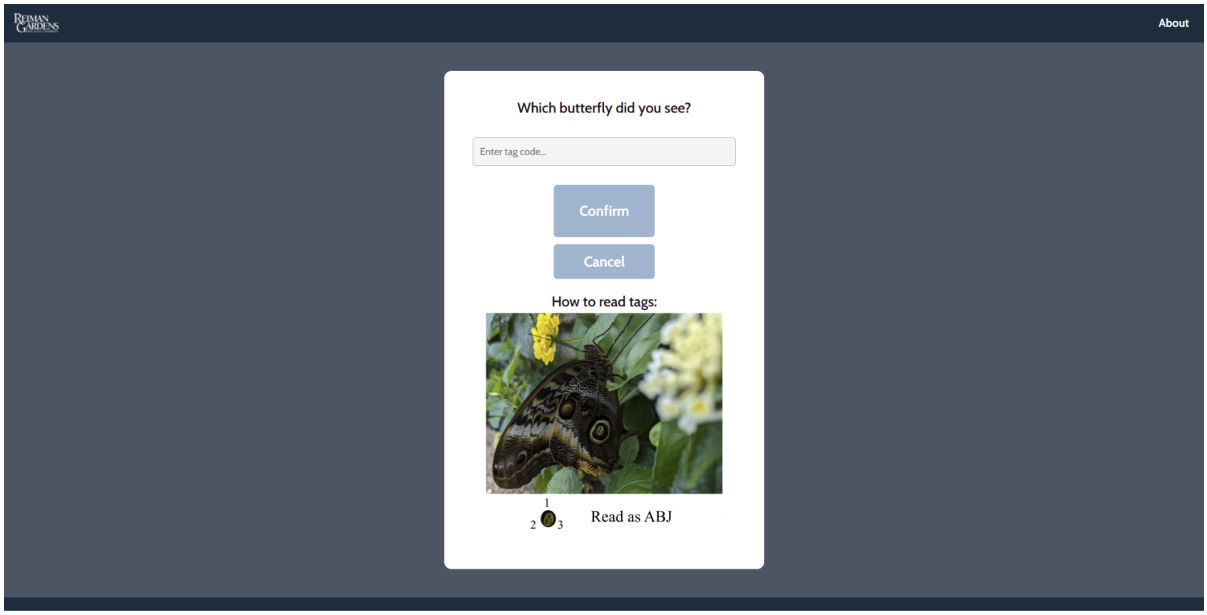


Figure 10.54

3. You will be met with a Success message, and then an image and some information about the butterfly that you spotted, including the Scientific and Common Name.

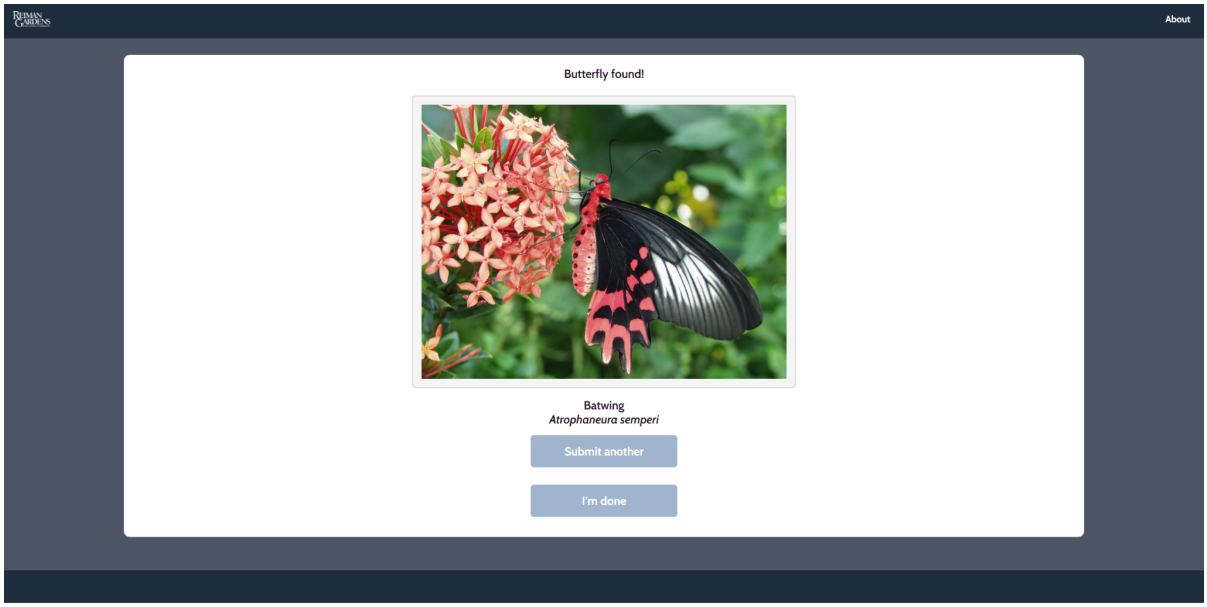


Figure 10.55

a. To continue spotting butterflies, click on the “Submit another” button.

b. If you are done submitting and are no longer interested in spotting butterflies, click on the “I’m done” button to return to the guest entry page.

10.1.20 Learn more about the butterfly exhibit

1. If you are a guest, follow the instructions on 10.1.18, Enter as a guest. If you are a docent, log in with your authenticated username and password. You will be auto redirected to the same landing page as the guest user. If you are a domain or super admin, log in with your credentials and press the “Enter a tagged butterfly” button.

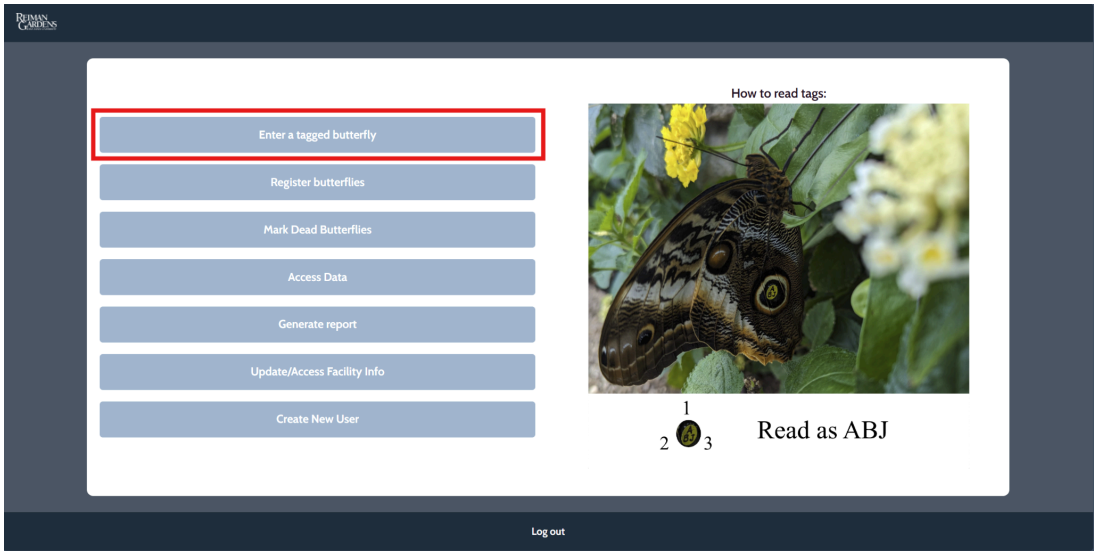


Figure 10.56

2. Click on the “About” button in the top right corner to access the domain information.

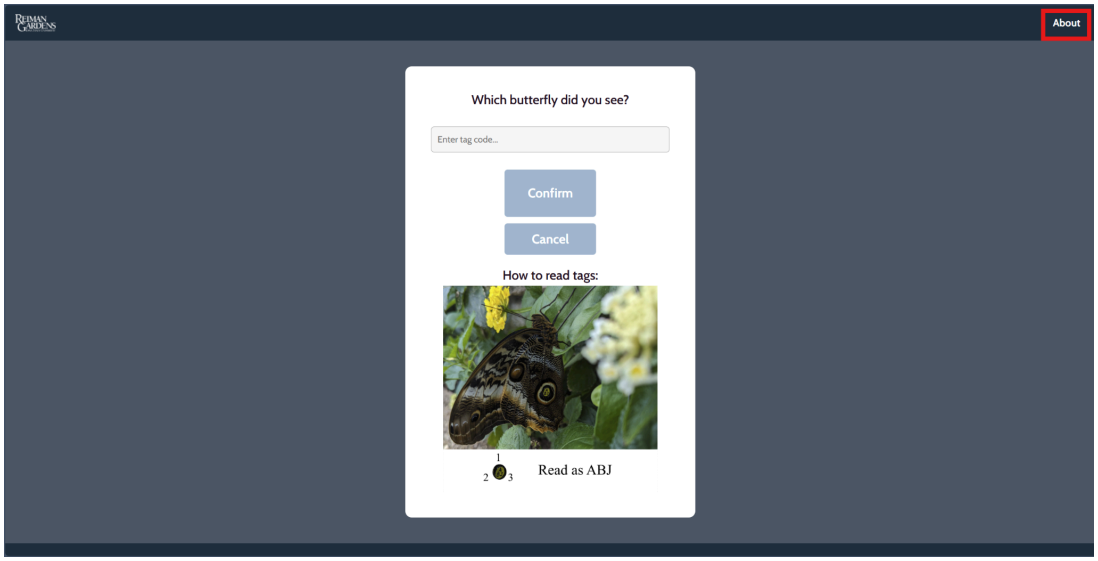


Figure 10.57

3. Here you will see the domain “about” information.

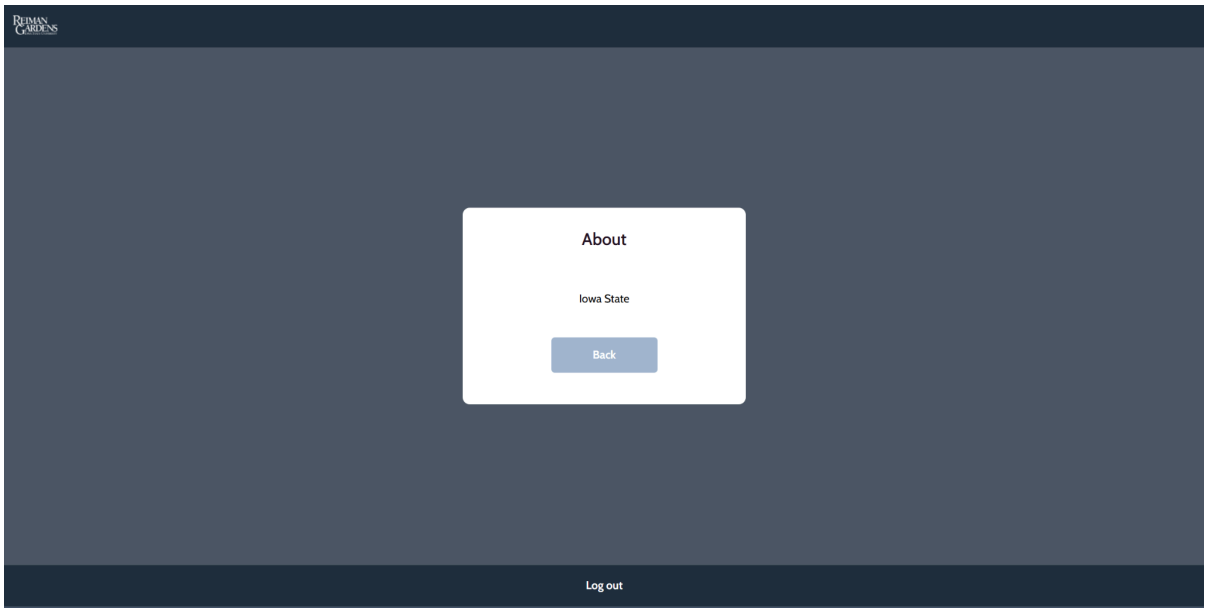


Figure 10.58

Appendix 2 – alternative/initial version of design

- Generating report version change
    - Originally, we were grabbing all of the butterfly data from the backend and filtering it on the frontend. However, we found that filtering the data on the backend is much quicker, and so we decided to stick with the backend filtering the data based on the request body and returning all of the data to be displayed.
- Butterfly tagging system
    - Originally, we were accounting for sticker color, sticker shape, letters/numbers, a single dot, or color of the letters/numbers on the sticker, and a customizable length of tag code. We were able to upgrade this system to allow domains to specify the type of characters they want to allow, use dots instead of letters/numbers, and a select number of colors that will allow the user to be able to read the actual color name. This upgraded system is simple and efficient, while providing more than enough customization to a domain's tagging system.
- Data display screen
    - In our original design, it was not able to handle colors or shapes of butterfly tags, which was an issue. We had to completely scrap the screen and use a

different design in order to effectively display the data. This was quite the setback, as displaying colors and shapes in a way that made sense to any user was complex. In the end, we were able to create one column that displayed the tag as it would be seen on the physical butterfly, which was an efficient use of space while still displaying the necessary tagging information.

Appendix 3 – Other considerations

- One important lesson that we learned is that the behaviour between production and development environments differs, and what works in one environment may not work in the other. It is important that in the future, we are able to test in a simulated production environment in order to ensure complete functionality when we deploy our web applications.

- Chunking large requests is very important, and making a request to process thousands of butterflies at one time is not practical. We went from doing a 290,000-line import request to chunking it into about 1,000 lines of JSON at a time.

- We learned that CORS policy errors happen a lot and arise a lot during local development. This caused a lot of issues with testing our communication between the frontend and the backend. However, after lots of trial and error, we were able to disable CORS and everything worked perfectly (we think).

- A variable named onlyDots kept on coming back within our code after merge requests, and we could never figure out why. It was not assigned to anything and just created errors within our JavaScript and unexpected behaviour. We have finally removed onlyDots from our code, until it one day inevitably returns.

- We had a bug that ran an infinite loop in our backend and fried our server. The infinite loop would trigger when a bad login request was sent. We thought we had the correct login credentials and were spamming the login button as a joke, which ended up killing the server. This happened a few times before we finally figured out why our backend kept getting really, really slow and then shutting off.

- Gitlab will auto-set your merge request to main no matter what branch you branch off of. This caused us to merge code into main when we didn't want to, multiple times, and create more stress than we needed.

- We decided to use MongoDB, which ended up being a mistake because it turns out we needed a relational database in order to run queries most efficiently. However, by the time we figured this out, it was too late, and we could not go back. We have made MongoDB into a relational database.

- When Charlie was going through the design document, he was commenting on changes we could make. He just didn't realize that he was commenting (ONCE- Charlie) on the prompts of the sections and not our actual responses. For example, he didn't like that the word 'salient' was being used (because he had never heard it before) and made a comment and changed it. We never wrote that word, it was in the section prompt.

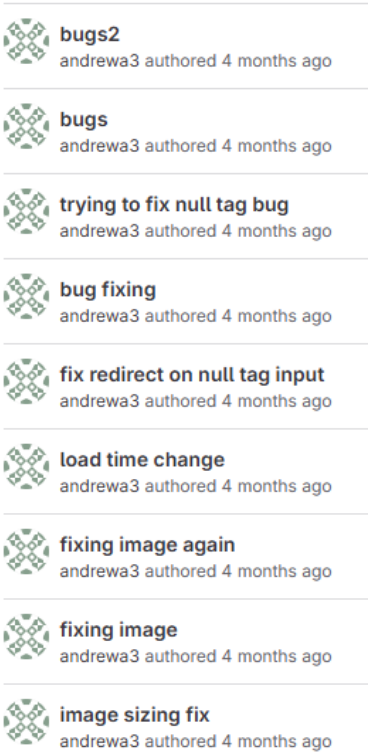
- Here are a few of our favorite commit messages-

- **LEBROOOOOOOOOON JAAAAAAAAAAAAAAAAAAAAAAAMES** - 2nd best commit of all time.
- **oh yeeaaah** - Oh no–accidentally merged into main
- **main is gone. main is deleted.** - Post merge to main, we deleted all the code in it.
- **FIRST TRY RAHHHHHHHHHHHHH** - Definitely was my first try.
- 

What was bro cooking?

Figure 10.59

- **big @ss update to all things back-end, MongoDB connected, Database Models...** - Big updates.
- **bug fixed, probably not in the most robust way but thats okay** - Yea no biggie.
- **removed duplicate foreground colors in main EZPZ CURRY FOR THREEEEEEE BAAAAANG** - Splash.
- **pushing to main because why not** - Who needs good coding practices?
- **Im perfect, formatted perfect** - He was feeling himself.

- **minor changes** - There were multiple push messages with this name, some were minor, some were major
- **changes** - idk what
- **Broken** - I don't think it worked
- 

  - This was all the same bug

Figure 10.60

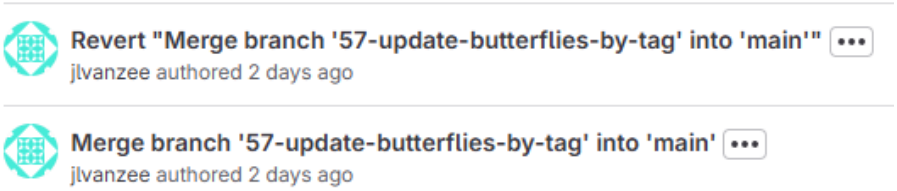
- **bing** - Alex literally just changed a constant to a variable
- **bug fixes with sizing** - you'll never guess who wrote this commit message
- 

Figure 10.61

- **I'm merging all over** - He merged

Appendix 4 – Code

- sd / sdmay25-03 · GitLab
    - Backend Folder GitLab
    - Frontend Folder GitLab

Appendix 5 – Team Contract

Team Members

- Alex Herting
- Andrew Ahrenkiel
- Carter Awbrey
- Charles Dougherty
- Jaret Van Zee

Required Skill Sets for Your Project

- Agile
- AWS Server Hosting
- CI/CD
- Cyber Security
- Database Management
- Data Security
- Database Design
- Gitlab
- Git
- Git Command Line Interface
- HTML/CSS/JS
- Java
- Java Spring
- JUnit Testing
- MongoDB
- Project Management
- SCRUM
- Social Engineering
- Software Architecture
- Software Testing

Skill Sets covered by the Team

Jaret Van Zee

- Agile
- CI/CD
- Cyber Security
- Data Security
- Database Management
- GitLab
- Git
- Git Command Line Interface
- HTML/CSS/JS
- Java
- Java Spring
- JUnit Testing
- MongoDB
- Project Management
- SCRUM
- Social Engineering
- Software Architecture

- Software Testing

Andrew Ahrenkiel

- Agile
- CI/CD
- Cyber Security
- Database Management
- Data Security
- Database Design
- GitLab, Git, Git CLI
- HTML/CSS/JS
- Java
- Java Spring
- JUnit Testing
- MongoDB
- Project Management
- SCRUM
- Social Engineering
- Software Architecture
- Software Testing

Charles Dougherty

- Agile
- Database Management
- Database Design
- Gitlab
- Git
- Git Command Line Interface
- HTML/CSS/JS
- Java
- Java Spring
- JUnit Testing
- MongoDB
- Project Management
- Software Architecture
- Software Testing

Alex Herting

- Agile
- Database Management
- Database Design
- Gitlab
- Git
- Git Command Line Interface
- HTML/CSS/JS
- Java
- JUnit Testing
- MongoDB
- Project Management
- SCRUM
- Social Engineering
- Software Architecture
- Software Testing

Carter Awbrey

- Agile
- AWS Server Hosting
- CI/CD
- Cyber Security
- Database Management
- Data Security
- Database Design
- Gitlab
- Git
- Git Command Line Interface
- HTML/CSS/JS
- Java
- Java Spring
- JUnit Testing
- MongoDB
- Project Management

- SCRUM
- Social Engineering
- Software Architecture
- Software Testing

Project Management Style Adopted by the Team

Agile

- GitLab issue board used to assign and track team progress
- Weekly group “stand-up”
- Project tickets/issues split amongst group members
- Issues identified and added to the backlog
- Biweekly client check-ins acting as product owner

Individual Project Management Roles

1) Alex Herting Frontend Developer

2) Andrew Ahrenkiel Full Stack Developer

3) Carter Awbrey Backend Lead

4) Charles Dougherty Frontend Developer

5) Jaret Van Zee Backend Developer

Team Contract

Team Members:

1) Alex Herting 2) Andrew Ahrenkiel

3) Carter Awbrey 4) Charles Dougherty

5) Jaret Van Zee

Team Procedures

(1) Regular team meetings outside of class will be done virtually over Discord. The regular team meeting will occur on Friday at 6 pm.

(2) The preferred method of communication for updates, questions, reminders, or general information will be through Discord chat or Discord voice call. If communication is needed for any inner conflicts or major issues, it will be done in person or via Discord video/voice call.

(3) Decisions will be made by a majority vote since there is an odd number of team members. Team members should be given a fair amount of time to discuss their choice and why they want that choice. Team members in the minority vote should comply with the decision made. Team members in the majority should see if they can compromise by including any qualities of the minority's decision.

(4) Carter Awbrey will be the official team auditor for team meetings. The backup team auditor will be Jaret Van Zee. The team auditor will be expected to take notes from each meeting. The notes should well document the current progress of each team member's work, any roadblocks, and planned work. All important information should also be documented in the auditor's notes, such as new meeting times, deadlines, and important workflow changes. The auditor should make the notes available no more than 1 day after the meeting to the rest of the team.

Participation Expectations

(1) All team members are expected to show up on time to any scheduled team meeting unless they communicate prior to the meeting. All team members are expected to participate whenever they can in a meeting. All team members should be actively listening whenever another team member is speaking. Conflict and discussion are okay in a team meeting, but members must respect each other's ideas, even if they do not agree with the idea being discussed.

(2) Team members will need to put their full effort into whatever work they are presenting to the rest of the team. All hard deadlines should be met ahead of time to review each other's work. Any teamwork will be divided equally among all team members as fairly as possible. A soft deadline will be determined, likely 2-3 days before the hard deadline, that which all team members should have their segment of work completed by.

(3) Team members are expected to regularly update their other team members. If any team member runs into a roadblock, they should notify their team members in no more than 24 hours. If a team member runs into any difficulties or roadblocks, a discussion or team meeting should be held in order to solve the issue.

(4) A team member is expected to work on their work or tasks for an average of three hours per week. Three hours per week is not a hard limit; this number could be higher or lower on a weekly basis. These three hours will not include any regularly scheduled team meetings.

Leadership

(1) These are the leadership roles that each team member fulfills:

Jaret Van Zee - Database Manager & Timeline Organizer

Carter Awbrey - Project Manager & Visionary

Alex Herting - Frontend Manager

Charles Dougherty - UX/UI Design Director

Andrew Ahrenkiel - Team Organization & Technical Design

(2) These are the strategies each team member will follow to support and guide the work of other team members.

Jaret Van Zee - I will ensure that all team members are getting assigned the work that is appropriate for them. I will also make sure that work is getting split fairly between the team members and that work is getting done on time.

Carter Awbrey - As the project manager, I will oversee the communication between members to ensure that deliverables are met and that our product meets the needs of our client. Additionally, I will work to address and mediate issues that may arise between team members. This doesn't limit my work to solely leadership-based contributions, but does include them.

Alex Herting - As the frontend manager, I will be responsible for creating tickets related to the front end and assigning them to members of the team. Checking in to see how far along we are on tasks and re-assigning tickets accordingly.

Charles Dougherty - It is essential that all parts of the team work collaboratively to create one whole product rather than pieces strung together. I will ensure that each person feels included and understands the goal of what is currently being worked on, and make sure that what has already been completed follows the requirements. I will also contribute to handling conflicts in code or design choices to create a better final product, even with differing ideas.

Andrew Ahrenkiel - As the Team Organization leader, I will ensure weekly meetings have applicable purposes and are productive. I will be responsible for the Gitlab issue board and tracking member progress. As the Technical Design Lead, I will ensure all code within the development branches is functional and appropriate; I will review the majority of code merges to ensure code quality.

(3) These are the strategies each team member will follow to recognize each other's work.

Jaret Van Zee - Keep to set soft and hard deadlines for work that needs to be completed. Will do regular check-ups on team members to ensure they are doing okay on their own work.

Carter Awbrey - Follow set team deadlines and communicate regularly and clearly with other team members.

Alex Herting - Setting appropriate deadlines and checking in weekly on the front-end tasks to make sure that we are meeting those deadlines.

Charles Dougherty - Follow deadlines, checking in on each area of the project and not just where I am currently working, checking GitLab commits and pushes, and discussing future improvements/changes.

Andrew Ahrenkiel - Gitlab issues, weekly check-ins, code commits and pushes, feature branches, etc.

Collaboration and Inclusion

(1) These are a list of skills, experience, expertise, and unique perspective that each team member brings to the team:

Jaret Van Zee - He is currently working as an IT intern. Jaret also has skills in back-end development and database management. Jaret can use his knowledge from his cybersecurity minor to bring new perspectives to the team and the overall security of the software being developed.

Carter Awbrey - Skills include server-side and cloud integration having worked with cloud products many times in the past. Additionally I have experience writing UI in HTML/CSS/JS or using React frameworks. I have professional experience writing backend/server software in .NET and Java as well as integrating that with frontend products.

Alex Herting - Skills include full-stack development, having experience with React framework and SQL for databases. I currently work as a software developer, which has given me experience with the software development process and managing large workloads. Have experience in JavaScript in web development.

Charles Dougherty - Experienced in working in teams to develop a strong final product. During my internships, I have worked in SQL and MongoDB databases and also developed many frontend applications. Finding new solutions to a problem is one of my strong suits as I enjoy finding new technologies and discovering what can be done within the limitations.

Andrew Ahrenkiel - Skills include full-stack development, particular experience with both React and Angular, along with Mongo, SQL, and Oracle. I can bring a unique viewpoint of the SDLC from my internship experience with Wells Fargo, having worked as a DevOps engineer

and a Fullstack Software Engineer. I also recently worked on a database migration project as part of my internship, which may provide me unique expertise in database design

(2) Throughout the semester we will keep track of our tasks using an issue board and milestones with deadlines that we would like to meet. We will appropriately assign tasks to each member such that everyone had an appropriate workload to complete for each milestone. We will establish a safe environment for curiosity and learning where members are not afraid to ask questions and feel comfortable having differing opinions. This promotes everyone to speak and voice their opinions on project issues and challenges that arose.

(3) We will first have a confidential 1-on-1 conversation with either a trusted team member, our advisor, or our client. We will ensure that any collaboration and inclusion issues are taken seriously and handled amongst the group.

Goal-Setting, Planning, and Execution

(1) These are the official goals that this team will strive to achieve this semester:

I. Learn how to work with each other as a team.
II. Build a good working relationship with our client(s) and supervisor
III. Continuously present high quality work as a team and individually to our professors, client(s), and supervisor. Ensure that all work completed meets its expected deadline.
IV. Regularly communicate with each other about how progress on our work.
V. Create a detailed and exact prototype of product we plan to build for our client in the following semester.

(2) Whenever new work is assigned to the team, a timeline should be created that the team will follow to make sure that the work is completed by the deadline. Each team member will be assigned work that best meets their expertise or interest. All work will be divided evenly to the best of our ability.

(3) During each team meeting, each team member will discuss what work they have completed, what work they plan to complete in the next few days, and if the member has run into any issues. The other team members will compare their current work progress to the expected timeline of progress to ensure that the team member is getting their allotted work complete.

Consequences for Not Adhering to Team Contract

(1) If any team member knowingly violates the team contract, the following procedures will be met.

First Infraction - A soft warning, the team member that committed the infraction will asked by another team member to not commit that infraction again

Second Infraction - A strong warning, one or two team members will directly say how the team member's infractions have affected the team and what they've done

Third Infraction - Official Team Meeting, an official team meeting will be made in order to discuss how this can stop in the future. A resolution must be reached and a remediation plan must be created in order for the team meeting to be successful

(2) If a team member is consistently committing infractions against the team after the warnings and team meetings. All team members except the team member committing the infractions will meet with the SE 491 professors to discuss a best possible solution for the rest of the solution. This may involve 491 professors directly talking with the team member committing the infractions, a specialized plan may be created for the team member, or possibly removal of the team member from the team.

****

a. I participated in formulating the standards, roles, and procedures as stated in this contract.
b. I understand that I am obligated to abide by these terms and conditions.
c. I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Andrew Ahrenkiel DATE: 09- 19 - 2024

2) Alex Herting DATE: 09 - 19 - 2024

3) Charles Dougherty DATE: 09- 19 - 2024

4) Carter Awbrey DATE: 09- 19 - 2024

5) Jaret Van Zee DATE: 09- 19- 2024