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 necessary functionality for the clients needs. We also need to ensure that they are in locations that would make sense to our client 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 have decided to use MongoDB as our database which is a very important decision to 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 applications availability. AWS offers high availability (99.999%) making it extremely reliable for our client and for the other users. AWS also allows for you to upgrade your plan at any time in order to account for a higher amount of traffic. This will allow 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

For deciding 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 5 considerations and why we chose to use/chose to not use them are as follows:

MongoDB (Using):

Pros

- Flexible data schema
  - Allows for easy changes in data collections
- Allows for scalability of data size without a tradeoff in performance
- High performance speeds
  - Fast querying speeds
  - Fast retrieval speeds
- Very little to no cost

Cons

- Does not support traditional SQL joins
  - Can lead to limitation in querying
- Potential redundant data
  - In turn increasing storage costs

MySQL (Not using):

Pros

- High performance speeds
    - Fast querying speeds
    - Fast retrieval speeds
- Built-in security features
- Supports vertical scalability

Cons

- Does not support horizontal scalability
    - i.e. adding traits to a data type
- Lack of schema flexibility
- Limited JSON support

Oracle (Not using):

Pros

- High performance speeds
    - Fast querying speeds
    - Fast retrieval speeds
- Multi-platform support
    - Available on Windows, Linux, Unix and more
- High scalability without performance tradeoffs

Cons

- High costs
- Resource intensive
    - Significant demand of CPU
- Built for enterprise applications

IBM DB2 (Not using):

Pros

- High performance speeds
    - Fast querying speeds
    - Fast retrieval speeds
- Supports XML and JSON formatted data types
- High scalability without performance tradeoffs

Cons

- High costs
- Complex setup and features in order to make the most of the application
- Low flexibility of data schemas

MariaDB (Not using):

Pros

- Open source and free to use
- Compatible with MySQL
- Flexible storage engines

Cons

- Lacks advanced features that can improve performance
- Low performance with large data sets
- 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	20	8	7.3
Oracle	9	9	6	3	8	7.1
IBM DB2	9	9	4	4	8	6.9
MariaDB	7	5	6	20	7	6.9w

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.

4.3 Proposed Design

4.3.1 Overview

We will follow a Model-View-Controller design pattern for our website. This means that there are three main components to the website. The first component is the view, which is the part of the website that a user will see and interact with. This part of the website must have a well-designed user interface and will need to take in user input safely. The second component is the model, which relates to all the data for the users, butterflies, and facilities that must be stored. We need to establish standards of what exact data will be stored, this will prevent any clashing of data or inconsistencies in the database. The final section controller, this part of the website, is responsible for moving any data from the database of information to the actual display on the website. The controller will need to input data and any data that the database provides in a secure and efficient manner.

4.3.2 Detailed Design and Visual(s)

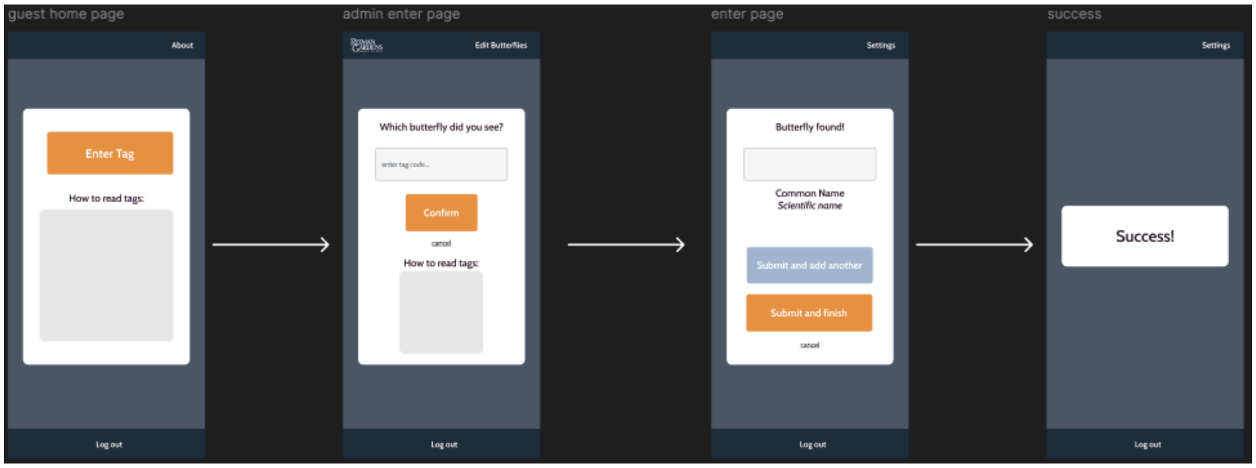
Front End

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 avoid performance issues and high device compatibility. 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 cannot be easily containerized and will be later implemented manually into the system. These views would include the database spreadsheet view and the admin home page setup view.

The code then needs to be optimized to resize correctly to mobile, tablet, and desktop views of the web app. For each view, this will be done quite differently since the containerization of objects in the view is quite different. 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 will need custom styling from developers to ensure proper function for all screen sizes.

For a more indepth example of how the app will work, watch the video demo at this link: https://youtube.com/shorts/4BnmfrQZxho



(Example of guest user experience)

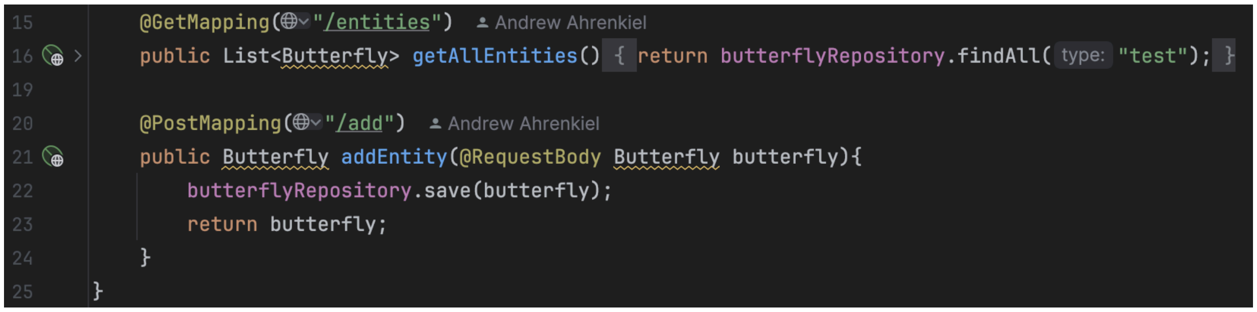
Another notable Frontend performance feature to be implemented is storing reporting information on the client side. As of right now a problem with the reporting is when searching the table or adding filters to generated reports, a new server request is made with every change. instead we can have all this filtering be done on the client side to avoid server lag and improve the overall performance of the system.

Implementation of report views for mobile devices are yet to be created. If the client desires to keep the table report for mobile devices then reporting views will be optimized for mobile and tablet devices. Ensuring these are easy to understand and use is up to developer implementation and will be constantly updated and corrected for improvement.

Backend

As mentioned above, Java Spring is the utility service that is being used for the backend of the application. There are two main purposes of the Backend, communicate data from the backend to be displayed to the user on the Frontend, and take user data from the Frontend and store it in the database. This is done by following a Get, Post, Put, Delete HTTP connection API, which is readily available through the Spring service.

Controller classes within the backend will be created based on views. Currently, All the Login pages will have an associated controller, the Generating Report page will have an associated controller, and the Tagging pages will have an associated controller. The image below shows an example of a Get and Post mapping implemented for testing within the controller.



```
    @GetMapping("/entities")
    public List<Butterfly> getAllEntities() { return butterflyRepository.findAll( type: "test"); }

    @PostMapping("/add")
    public Butterfly addEntity(@RequestBody Butterfly butterfly){
        butterflyRepository.save(butterfly);
        return butterfly;
    }
}
```

(Example test code)

Each of the following table entries will have its own implementation of a request mapping. All requests will use JSON for communication as that is the form the data will be stored in. Request code can be written in multiple ways and will be up to the developer's interpretation for implementation.

Get	Post	Put	Delete
Get Facility Theme	Login Request	UpdateFacility 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 specific facility	Create Facilities and Facility Admin Accounts	Update Admin Account password or information	Delete Facility and Facility Information
Get all butterfly	Add a new Butterfly	Update a butterfly	Delete/Remove Butterfly

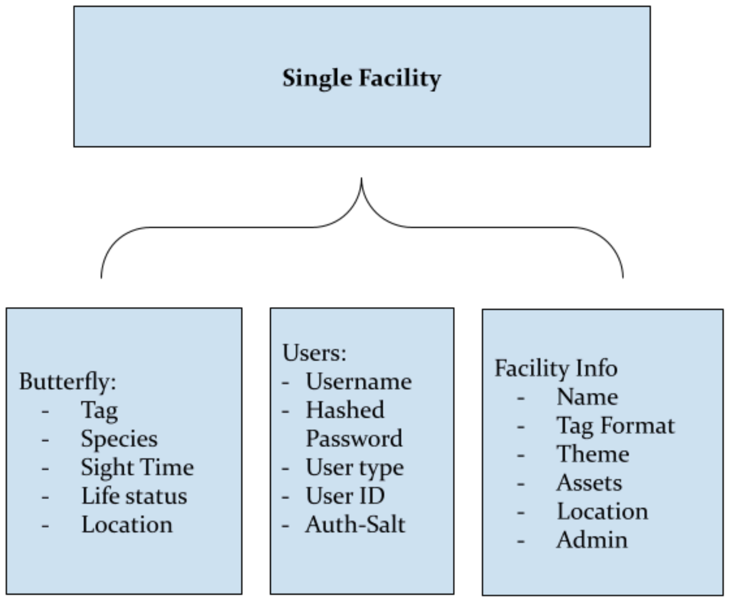
information with specified filters	Species	sightings information	Species

Requests will use JSON request bodies when needed rather than parameter variables to keep api url simple and easy to understand. Response bodies will also be given in JSON so data will not have to be altered after retrieving it from the database. Requests should be reused whenever possible to avoid duplicate requests in different controller classes.

Database

MongoDB is the chosen service for our database, with the reasoning shown in the above sections. I will describe the high-level database collection schema and how it will hold information for our users. It's notable to mention that MongoDB is a non-relational database meaning it does not utilize tables to hold information. Rather, MongoDB stores data in collections; within collections are objects that store data in a JSON format.

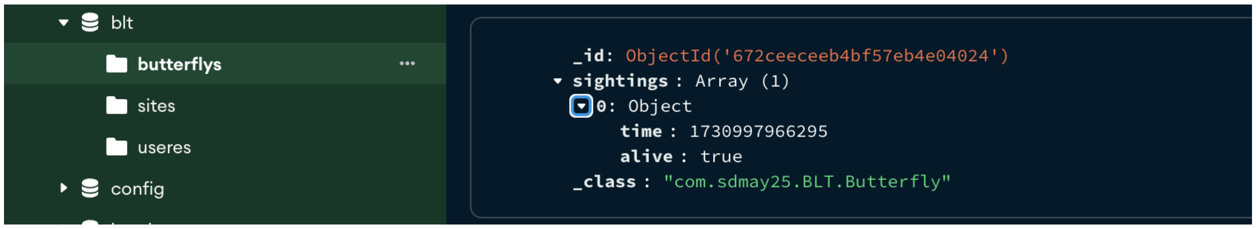
To maximize performance for each facility, we have chosen to create a collection for each facility that will be utilizing the web app. The following model resembles how the database collections will be organized.



(Example of Database collection hierarchy)

Each facility will have its hierarchy of collections with the above fields being included in each collection object. Separating the facility butterflies from each other makes it much easier to give reporting information based on that individual facility. This means only their butterfly data will be parsed for filtering and returned to the user.

In the special case, a super admin will request butterfly reporting information from multiple facilities; this is the only time a join condition will be implemented. Since different facilities may have the same tagging method, it's essential to include the location of each butterfly directly within its database object entry. This means even when multiple facility butterfly collections are joined together and presented, Butterfly entries can still be distinguished by location.



```
_id: ObjectId('672ceeceeb4bf57eb4e04024')
▾ sightings : Array (1)
  0: Object
      time : 1730997966295
      alive : true
_class : "com.sdmay25.BLT.Butterfly"
```

For a more high-level and easy-to-understand model of the design, please refer to the following document:

Detailed Design and Visuals

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 enables users to easily input sightings, tagging location, time, and other details. These inputs are stored in a database for streamlined analysis and retrieval.

Additionally, the app includes a robust user management system, allowing for varying access levels based on user roles. This ensures that different users, from researchers to citizen scientists, have tailored access to the database and app features. For example, while a researcher 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 Concern and Development

Our current design is on track to meet all project requirements and user needs. So far, any challenges we've encountered have been manageable with our chosen tools and design approach, which offer the flexibility and scalability needed for our application. The extensibility of our tools has allowed us to adapt smoothly, and we see no need for major changes at this stage.

At present, we have no significant concerns about the development process, and thus no specific issues to address. However, as we move forward, we'll continue monitoring for any potential areas needing refinement and remain open to feedback from clients, TAs, and faculty advisers to ensure our design remains aligned with expectations.

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.

4.5 Design Analysis

So far, we've successfully translated our frontend design concepts from Figma into HTML, which is now fully viewable and loadable. However, we have yet to implement the backend functionality and connect the pages, which will primarily involve JavaScript. In the future, we will need to link the frontend to the backend APIs, which are being developed concurrently. On the backend, we've made progress on implementing the APIs and connecting the database to support our needs. We have completed basic functionality for butterfly tagging and sightings, but still need to implement user authentication, user management, and additional API features to complement the tagging and sightings system. Looking ahead, we plan to focus on finalizing these integrations and ensure the full system works seamlessly. While the overall design is feasible, we are addressing key functionalities in parallel to avoid any delays in the user experience or data management.