

| **Sponsors:** | Northern Arizona UniversityUSDAUS Forest Service |
| --- | --- |
| **Clients:** | Dr. Jamie S. Sanderlin, Dr. Ana Miller-ter Kuile, and Dr. Kiona Ogle |
| **Mentor:** | Scott LaRocca |
| **Team Members:** | Alyssa Ortiz, Andrew Ortega, Payton Watts, and Tyler Chapp |

**Software Design Document, version 1.0**

**February 7, 2025**

## Table of Contents

[1. Introduction 3](#_mclgj8nfy5s7)

[2. Implementation Overview 3](#_fu6qo364hzrt)

[3. Architectural Overview 4](#_keqme67j2j22)

[4. Module and Interface Descriptions 7](#_kf1bxh4wahfb)

[5. Implementation Plan 12](#_kc0azvkj8rwx)

[6. Conclusion 14](#_ekcznk2ezi5q)

## **Introduction**

In the United States, forests play a critical role in supporting both the economy and the environment. With approximately 40 percent of the country covered by farmland and over a third of its land area forested, it is evident that the health of these ecosystems is vital to agricultural productivity, economic stability, and overall ecological balance. Despite the current extent of forested and agricultural land, destructive factors, such as wildfires, are increasing due to climate change and other harmful influences. These factors are contributing to significant damage in forests, raising serious concerns about the long-term effects on both the environment and society.

 Thankfully, nature provides its own detailed litmus test for the health of ecosystems- that being birds. Due to the many interactions they have with an ecosystem(natural pest control, pollination, seed dispersal), birds make a great indicator of the ecological integrity and the current health of the wildlife in an area. Using prior trends and data, the presence of birds in an ecosystem can be predicted, and thus by extension, the future of a recently-affected ecosystem itself can also be predicted. If birds are not predicted to be returning to an ecosystem that just faced a natural disaster, it could be a sign that the ecological integrity of that area is in danger; while on the other hand, an ecosystem that shows high levels of predicted bird presence is much more likely to be on the path to recovery from a recent catastrophic event.

 The key to accessing and interpreting this data lies in the people collecting it and making the predictions, which happens to be our sponsor, the USDA Forest Service, in particular Dr. Jamie Sanderlin, Dr. Ana Miller-ter Kuile, and Dr. Kiona Ogle. The latter two of our sponsors perform related research at NAU, with Dr. Ogle in particular representing the School of Informatics, Computing, and Cyber System’s Environmental and Ecological Informatics program. Together, they have collected over ten years of bird data from the forests of the Western United States (excluding California), and used it in combination with high-level statistics to make a database of predicted bird presence data across various states. Our team’s purpose is to take these large datasets and allow users to efficiently query it and retrieve data and/or visuals, so that researchers, such as our sponsors, can effectively communicate their concerns with important parties. These datasets and visuals will coincide with each other in informing researchers, or the general public, of the health of a given area(based on the volume of birds). Using this information, researchers can make educated decisions about ecosystems and determine the course of action required so that some natural disasters can be averted.

## Implementation Overview

Our desired implementation relies on the data sets provided by the US Forest Service. Using this data, we will create a website that allows government officials and the general public to interact with the data. There will be tabs that give different information, based on the tab, with an emphasis on the map tool. This tool will interpret data and display it accordingly. Users can choose what data aspects they want to view and modify the map to display these aspects. That is to say, the map will allow for multiple filters to be enabled on it.

In order to get these functions to work, we will use a combination of Python, Rasterio, Folium, Bootstrap, Django, and Amazon Web Services. Python is our main coding language we will utilize for most of the project. It will be used with Rasterio and Folium to access, read, and display the data for the map. We will use the Python-based Django for our web framework to set up our website and ensure that it is developed properly and professionally. Bootstrap, a CSS and javascript library, will be used to style the website so that it is user friendly and aesthetically pleasing. Our main source of hosting is Amazon Web Services, using an EC2 server.

## Architectural Overview

**Overview:**

 The system architecture of the project is deep and multifaceted, with a wide array of back and front end technologies involved. Primarily, the main components at play are the user interface, the querying and mapping/visualization module, the application logic, the database layer, and lastly, the hosting and version control management.

**Visualization:**

(See next page)



**User Interface:**

 Starting at the most high level section, there is the user interface. This will be the face of the project, and what makes the application specifically a web application. The front end will be presented to users by the website, offering a user-friendly user interface that will help them achieve their goals. The main area in which said user goals will be fulfilled will be the data querying page of the website, in which users will be presented a variety of options for getting and presenting data. Users will be able to query data through our application by interacting with either checkboxes, drop-downs, search bars/input fields, or a combination of all of these. Through these methods, the user will be able to specify an area in which to receive the bird presence data from, and then subsequently choose to have it returned to them in file or map form. These requests will primarily be fulfilled by HTTPS and the Django web framework, while the actual interactions and appearance will be handled by JavaScript, and HTML/CSS respectively.

**Application Logic:**

 The aforementioned requests will be taken in by HTTPS up front and passed on to be processed by Django, which will be essentially “waiting” and interwoven into the front end. For whichever action the user is trying to take with regards to accessing and processing the data, an appropriate Django view will be there to receive it. This view will then execute Python code in combination with the related Django model(s) needed to fulfill these requests. These models are Django’s interface with the database and its elements, and are a powerful tool in user input/output between front and back ends.

**Querying/Visualization:**

A deeper look into these views and the Python code they contain yields the next layer of architecture- the querying, mapping/visualization component. The data querying section of this component deals with retrieving the bird presence prediction data for the area requested by the user. This area of the code will be working incredibly close with the MySQL server and its relevant tables. If a file of specific data was all the user requested, once querying is done, a CSV file will be promptly sent back to the front end. However, if a map is requested, this queried data will then be passed on to the mapping and visualization section. This part of the code will convert the retrieved data into raster format, through the Rasterio Python library. This raster data will then be processed by the Folium library into a map, with specific layers as requested by the user. At this point, the visualized data will then be handed back up to the front end for user viewing and interaction.

**Database Layer:**

 The data that the querying and therefore mapping/visualization will be drawing from will be retrieved from the next layer in the software’s architecture- the database layer. This will be the aforementioned MySQL server, along with Django’s database interfacing capabilities. The three main tables the querying section will be dealing with are ones for species data, grid area data, and the predicted results themselves. Should these tables be empty or in need of additional/updated data, a custom Django command will be available to automatically populate one of these tables from an input file of the proper format.

**Hosting/Version Control:**

 Lastly, there is the hosting and version control layer, arguably as much of the software’s backbone as the database layer. This server hosting will be handled through AWS, Apache and other resources, allowing a flexible and responsive experience for both users and developers alike. Version control will be another part of ensuring a cohesive and dynamic experience for developers, through Github. It will ensure that everyone will be developing the same copy of the software, and avoid incongruous changes.

**Conclusion:**

 With its wide variety of different components, the project's architecture also draws inspiration from many different styles. The overall design is very strongly aligned with layered architecture, starting at the high-level, user interface section, and then progressively getting into lower level software and paradigms as the architecture gets deeper. Also influential in the project are the client-server and event-driven architectural styles, clearly visible in how the user interacts with the website to request specific data from the back end, which is then found, processed, and returned to the user back on their side. And lastly, none of this particular architecture setup for the application would be possible without Django, which itself follows the model-template-views (MTV) architecture, where a relational database is the model, the template being the website, and views being the HTTP interactions of the website and their processing.

##

## Module and Interface Descriptions

With a clear understanding of the overall architecture of the system, there are different aspects to the project that work cohesively to ensure the product is efficient and accurate. They are a simple user interface, mapping/visualization module, application logic, database layer, and hosting/version control.

### 4.1 User Interface

 The user interface consists of multiple parts. There are input fields, legends, and interactive filters for the user to alter the display data. The input fields focus on the desired area that the user wants the information to reflect (i.e., state or region). This is accessible because of the map data being generated interactively. The legend will change with the user. If the user zooms in, the proportions displayed in the legend will alter accordingly. If the user selects certain filters, the legend will interpret these filters and display the required information that helps give a better understanding of the map. Based on the region selected, a raster displaying bird abundance data will be generated. A drop-down menu will help users gain a better understanding of the area that they are looking at, allowing them to view certain filters on the map. This is accomplished through the use of method calls. One such method, request.method(), takes in all the bird species from the CSV files and displays them all in a drop-down menu for the user to select. Another method will be used that will help focus on the sizing of the grid for our map, to ensure that accurate zooming in and out features and an appropriate legend are compatible. Also, a feature will be used to export the map displayed and turn it into a standard picture for users to view on their own, or extract as a CSV file.



### 4.2 Mapping and Visualization Module

 The mapping and visualization module is responsible for rendering geospatial data interactively and dynamically, ensuring users can efficiently explore spatial datasets. To achieve this, the module incorporates methods such as a dynamic map update routine that accepts a user-defined bounding box, fetches the corresponding geospatial data through an integrated data fetching method, and then renders this data using a dedicated rendering process. In addition, layer control methods allow users to toggle between various geospatial overlays, including vector features, heatmaps, and satellite imagery, while providing options to update individual layers or multiple layers simultaneously. A raster generation service is also included, which preprocesses raw geospatial data and converts it into GeoTIFF format through a structured raster conversion method, thereby optimizing the visualization and storage of high-resolution imagery.



 The system architecture for this module consists of several interconnected components, each supported by specific methods that facilitate efficient data visualization. The dynamic map update component continuously triggers a method that retrieves geospatial data for a defined bounding box, ensuring that spatial features are rendered in real-time. Complementary layer control methods manage the display of multiple map layers, allowing users to customize the information visible on the map. The raster generator, which forms another integral part of the system, employs a sequence of methods, beginning with data preprocessing and followed by a conversion process, to transform raw geospatial datasets into GeoTIFF files using advanced geospatial libraries. Together, these methods ensure a seamless interaction between the user interface and the underlying geospatial data sources.

 The public interface of the module exposes essential services for comprehensive geospatial visualization by integrating these methods into a cohesive user experience. For example, a dynamic map update function allows users to define a bounding box, which then activates the associated data fetching and rendering methods to continuously update the displayed features. Simultaneously, the layer control interface offers methods to dynamically enable or disable specific layers based on user preference. Furthermore, the raster generator service abstracts the complexities of geospatial data processing by combining preprocessing and raster conversion methods to generate GeoTIFF files optimized for high-resolution display and efficient storage. These interface elements collectively ensure that the mapping and visualization module remains flexible, efficient, and well-integrated within the broader system architecture.

### 4.3 Application Logic

The application logic layer largely consists of Django framework. This includes models for interfacing with the database, views for processing requests and publishing responses to client webpages, and custom URL patterns for easy and descriptive links. The application layer interacts with and connects the database to the viewable maps and data displayed to the website user. With Django, our application uses an Object-Relational Mapping (ORM) Library to query data related to our database. Using an ORM query handler makes our code easier to read, thus reducing the chance for errors or redundant code.

The views handle GET and POST requests to and from our website, using req() and res(). The views import all of the defined URLs from a separate URL file, and use these definitions to ensure that all GET and POST requests are standardized and always going to the same place. Each view takes in a request, and depending on the request, will return the appropriate response. Django has built in authentication and SQL-injection safeguards.

The data processing module plays a key role in transforming queried data into a format suitable for visualization. This module retrieves relevant spatial and non-spatial data from the database and structures it into GeoJSON format, which is easily interpretable by raster.io in the visualization module. By standardizing the data output, the processing module ensures seamless integration with the front-end mapping libraries, allowing for efficient rendering of geospatial information. This approach enhances usability and scalability of the application while maintaining consistency in data representation.

### 4.4 Database Layer

As described in the architectural overview section, the database layer, along with hosting/version control, are the backbone of this project’s software architecture. This is a data-driven project, and therefore having a well-developed and accessible database is incredibly important. The database will run on a MySQL server, which is naturally a very intuitive and user-friendly database system. Users requesting data or visualization data will interact with the appropriate Django view(s), which will then, through their application logic, consult the Django models system. Each model in Django is representative of a table in the connected database, and allows for Django to easily interface with it. From there on, Django will be able to fetch and process the database elements as needed, and then return them to the front end once finished.



In terms of the public interface of the database layer component, and the services it provides, it is the essential center in which all data retrieval and processing revolves around. First and foremost, the database needs to be able to be filled with elements, which is done using the auto-population command. Once a database is fully populated, it is ready for access by the application. The user’s requests rely on Django being able to successfully access the database through its models and retrieve elements from its tables. Django will then pass this data on to its views, to be processed into a file and/or map by the relevant Python code.

### 4.5 Hosting & Version Control

The hosting and version control module is responsible for the automated deployment and testing of the website and its code. The website is hosted on an Amazon Web Services EC2 server and utilizes Docker to deploy the live site. Docker ensures that the deployment environment is consistent no matter what platform the website is launched from, making it the best option. Automated testing is performed via GitHub Actions, any time there is a new commit to the main code branch. Github is also used for version control, where branch protections, pull requests, and version history ensure the integrity of the codebase while allowing for efficient workflows.



GitHub interfaces with the EC2 server by using push and pull requests to update the code hosted on the server. Simultaneously, it will also connect with the project's webhooks to notify the team of changes. It will run the automated tests to make sure everything works, then push the code to the server. When the server updates, Docker will tear down the old build of the webpage, taking it offline temporarily. It will then rebuild the application using the updated code, and redeploy the website.

## Implementation Plan

 This implementation plan outlines the development timeline for our project, detailing key milestones, deliverables, and responsibilities. The project follows a structured approach, beginning with initial setup and planning, followed by iterative implementation, testing, and final deployment.

The first three weeks focus on setting up the project infrastructure, including configuring the hosting environment, establishing version control, designing the database schema, and creating initial UI wireframes. The development phase spans weeks 4-9, where the core system components-user interface, database, and application logic-are implemented in parallel. A significant focus is placed on integrating geospatial mapping features, building out the API, and ensuring efficient data handling.

By weeks 10-12, full-stack integration is emphasized, ensuring seamless communication between components. This phase also includes optimization efforts such as query performance tuning, database optimization, and caching to improve system responsiveness. Comprehensive testing such as unit, integration, and user testing will be conducted to refine functionality and address potential issues.

The final three weeks are dedicated to feature refinements, user acceptance testing, deployment, and final documentation. Deliverables include the final project report, user manual, and fully operational application deployed to Amazon Web Services. The team will adhere to a parallel development approach to maximize efficiency and ensure timely completion of all milestones.

 Each team member plays a crucial role in ensuring the successful development of the project. Andy and Alyssa are responsible for designing and implementing the user interface, focusing on creating an intuitive and visually appealing experience. Tyler and Andy lead the mapping and visualization efforts, integrating geospatial features to enhance data representation. Payton and Tyler handle setting up application logic, ensuring that the core functionalities and system processes run efficiently. Alyssa and Payton collaborate on database development, structuring, and optimizing data storage for seamless performance. Hosting responsibilities fall under Alyssa, who will configure and maintain the cloud infrastructure to ensure reliability and scalability. Version control is a shared responsibility among all team members, ensuring smooth collaboration, efficient tracking of changes, and maintaining code integrity throughout development.

### 5.1 Team Member Responsibilities

| **Module** | **Team Member(s) Responsible** |
| --- | --- |
| User Interface | Andy, Alyssa |
| Mapping & Visualization | Tyler, Andy |
| Application Logic | Payton, Tyler |
| Database Development | Alyssa, Payton |
| Hosting | Alyssa |
| Version Control | All Team Members |

### 5.2 Project Implementation Timeline - Gantt Chart

## Conclusion

Climate change is affecting the world, more specifically, the surrounding region. Wildfires are on the rise, destroying entire ecosystems as they ravage the land. Researchers will not be able to determine the health of an ecosystem as easily without our tool. With the use of captured bird data, this project will help to determine the healthiness of ecosystems. The FireFlight project will incorporate several functionality requirements to ensure a productive and convenient tool for researchers, or anyone interested in learning about climate change. These requirements include a user interface that allows users to interact with the data in the form of an interactive map. The site will have backend data called and interpreted for accurate results to display in the form of a heatmap. It will access this data through the use of a MySQL server, ensuring that the data is secure. We have outlined the tentative deadlines throughout the project to ensure that the team makes steady progress on the project and that the project is fully functional by the Undergraduate Symposium. The team is confident in its ability to deliver a successful tool that can help monitor bird populations to estimate the health of an ecosystem for government and public use.