

| **Sponsors:** | Northern Arizona University  USDA  US 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 Testing Plan**

**April 4, 2025**

# Table of Contents

[**1. Introduction 3**](#_m70flfj03h6b)

[**2. Unit Testing 4**](#_qlfcj87hveou)

[**3. Integration Testing 8**](#_tez6lxqz6nne)

[**4. Usability/End-User Testing 10**](#_ge7cc9glnnaa)

[**5. Conclusion 12**](#_gqlnxe7wdz5m)

# 1. Introduction

Functional software is a necessity when creating a product that utilizes large amounts of data to display information, which is the teams’ main goal for this project. At the start of the process ensuring that the system is successful, the team will run several unit tests. These unit tests will make sure that each section of code runs properly- this will be achieved by using built-in functionality within our system. The team chose Django as the main “backbone” of the project, and it has the ability to run tests in a file to verify the functionality of each function. While running, the tests will also alert the team if there was an error, that way changes can be made according to the appropriate code to rectify the error. To scale up, tests regarding the integration of these sections, referred to as integration testing, will reach its goal if each separate section of code is capable of working cohesively with other sections of code to produce the desired outcome. These integration tests will be conducted to confirm that different aspects of the backend are capable of operating with the front end. This will be done by verifying that CSV data is able to be properly read, and that the read data can be displayed appropriately. This outcome being a functional website that is able to produce bird data on a map, and a website that is capable of being properly navigated.

End-user testing is required to guarantee that the website is as user friendly as possible, as the team intends it to be. To administer end-user tests, people who are not affiliated with the project will be asked to participate in a supervised questionnaire, where they will follow the provided instructions- resulting in questions being asked at the end of the process regarding the instructions and the site as a whole. These responses can give additional insight for the team and aid in potential additional functionality for the site. While these tests may seem rigorous and extensive, they are absolutely necessary. The team wants to be able to produce a product that can be used by our clients, and eventually the public. These tests will ensure that both the front-end and the back-end of the project work as intended, both separately and together. To begin these tests, we want to take a closer look into what a unit test would entail.

# 2. Unit Testing

## Overview:

To quote Amazon AWS, “Unit testing is the process where you test the smallest functional unit of code. Software testing helps ensure code quality, and it's an integral part of software development”, and that the “best practice [is] to write software as small, functional units then write a unit test for each code unit”. In the modern and professional software/project development processes of today, unit testing is considered more of a necessity than not. Django’s official documentation says that “when testing, more is better”, and that building too many tests is fine, as for the most part “you can write a test once and then forget about it. It will continue performing its useful function as you continue to develop your program”. Unit testing seeks to cast a wide net out across a software project, and see what issues it catches, with the overall mantra behind it being “better safe than sorry”. The end goal of all this is to allow for a well-defined set of tests developers can run after updating their code, to find any newly introduced bugs and errors. If a test case fails, they will be able to easily consult the conditions under which that test fails, and therefore be able to pinpoint where exactly the new error originated from.

The way this team is able to utilize testing is through the same means we’ve accomplished the majority of this project and its functionality- Django. As alluded to above, Django’s official documentation has a tutorial section entirely dedicated to running tests on your newly-built application, and the system as a whole is very friendly to testing. Django expects tests to be set up and defined within a specific “tests.py” file, placed in the main folder of the app. At the beginning of this folder, the necessary libraries are imported. At the bare minimum, the TestCase class from the django.test library is needed. This is the main building block for each unit. However, other libraries, functions, and classes are almost inevitably needed. One that we make use of is the call\_command function from the django.core.management library, which we utilize to be able to test the custom Django commands we’ve made. Another import that we make use of is essential for any querying/modifying of the database. This, of course, is importing our various models from our models file. An integral import we also implement is the csv library, which allows us to easily write to csv files, which is needed for making files to test certain units with. Used in similar circumstances is the StringIO object from the io library. This object gives us the ability to read in the data from various output streams, primarily stderr. These are just some of the many imports we make use of in our testing operations, but all of them serve a similar goal- to allow us to accurately create the conditions needed for our tests to be realistic and reliable.

In a massive project, unit testing can very likely involve dozens of different units, especially if the codespace is thoroughly modularized. However, for our team’s project, we’ve decided to focus on just a couple specific units in particular, mostly due to the size of each respective unit, in regards to function and code length. One of the units consists of the program’s database auto-population command. This custom Django command needs to be thoroughly tested, as to ensure that incorrect files and data are not accepted into the system and its database. Therefore, the command will need to be tested to see if incorrect file names and contents will be properly rejected. While still in the realm of custom commands, the commands built for generating the heat map will need to be another unit. It needs to be assured that these commands can successfully retrieve data from the database, read said data, and generate map data accordingly. Essential to the section of our project between back end and front ends, each of our views must be tested as well. This includes our views for grabbing and interacting with the HTML files, along with generation of maps, querying the database, downloading data, and so on. Last but not least, we must test the frontend. This consists of testing page loading, buttons, and the various frontend-view interactions. If any of these units are not set up to be tested extensively, we risk putting out an application that lacks a real cohesion amongst its various parts, and one that risks having errors lying in wait after update.

## CSV to Database Unit:

|  | **Equivalence Partitioning** |  |
| --- | --- | --- |
| **Invalid** | **Valid** | **Invalid** |
| **Filename Incorrect** | **Filename Correct,**  **Headers Correct, and Data Correct** | **Headers or Data Incorrect** |
|  | **Boundary Value Analysis** |  |
| **R3\_EnchantedCircle\_birdspecies, birdSpecies.py, abcdefghijklmnopqrstuvwxyz, etc.** | **R3\_EnchantedCircle\_birdspecies.csv,**  **["speciesID","species","birdcode"],**  **[1,American Crow,AMCR]** | **(Any headers that don’t line up with the format of the three main CSV types), a string where an int was expected, etc.** |

This unit focuses on one of the most important building blocks of the program, the Django custom command that takes in a CSV file that corresponds to one of the three models, parses it, and populates/adds its data to the application’s database. To work, a valid file name/path must be passed in as a command argument. Within that file, first, the headers must first correspond to one of the three file types- species, grid, or results. If the headers are correct, then last but not least, the data must be properly formatted. For example, if a numerical bird species ID is expected, and a string is in the expected place instead, an error must be raised. There are six tests, each testing if the command will respond properly to an incorrect input. The first test tests a file with incorrect headers, and the second test tests having an incomplete set of header fields. The next three tests test bad data in each of the three file types. The sixth test passes in a Python file instead of a CSV, and the last test inputs a gibberish string instead of a real file name.

## Mapping and Visualization Module:

|  | **Equivalence Partitioning** |  |
| --- | --- | --- |
| **Invalid** | **Valid** | **Invalid** |
| **Missing/Malformed Dataset and Generation** | **Valid Dataset** | **Malfunctioning Interactive Elements** |
|  | **Boundary Value Analysis** |  |
| **Using a dataset that doesn’t exist, or has corrupted or incomplete data. Generating a non-colorblind-friendly map.** | **Using complete, correct data from an existing file to generate a colorblind-friendly map. Interactive elements only appear when the map is in fullscreen mode.** | **Back button and other interactive elements appear while the map is embedded.** |

Unit testing for this module focuses on ensuring that each function in our mapping pipeline, ranging from CSV data parsing to the rendering of the final Folium map, is reliable and robust. For instance, the custom Django command includes a method called create\_heatmap\_raster that processes grid coordinates from the database and posterior median values from CSV files. Unit tests should supply both valid and intentionally malformed datasets to this function to verify it correctly calculates raster dimensions, applies Gaussian smoothing, and writes the output GeoTIFF with expected bounds. Additionally, the conversion function convert\_geotiff\_to\_png can be tested with controlled input data to ensure the normalization and custom colormap are correctly applied, producing a PNG that visually matches expectations. Similarly, components that handle the interactive elements, such as the custom BackButton MacroElement in the Folium map, should be isolated and tested to confirm that they render only when not in embed mode. By mocking database queries and file I/O operations, these tests validate the modular functionality of data ingestion, processing, and visualization without interference from external dependencies.

These unit tests play a crucial role in ensuring that any changes to the codebase do not inadvertently break the functionality of the mapping pipeline. They enable rapid identification and resolution of issues, thus facilitating continuous integration and deployment. Moreover, maintaining a comprehensive suite of unit tests fosters developer confidence and supports the iterative enhancement of the module as new features and optimizations are introduced.

## Views Module:

|  | **Equivalence Partitioning** |  |
| --- | --- | --- |
| **Invalid** | **Valid** | **Invalid** |
| **Incorrect URL** | **Correct Calling and Usage of View** | **Invalid Value Passed In** |
|  | **Boundary Value Analysis**  **(Example Used: Query View)** |  |
| **website-url/query/species** | **website-url/species/query** | **website-url/invalidmodel/query** |

Unit testing the views entails testing that the correct pages are loaded when requested, and that all elements of these pages function properly. For example, testing that the navigation bar has all of the correct links and loads to the expected page is essential- as the website would not function properly otherwise. We will also test that our implemented 404 error page is displayed correctly, and in appropriate situations.

The most important view to be tested is the map view, as it is directly involved in the CSV and map generation for the website. For this view, testing invalid object requests is critical, so that we can ensure that the system responds correctly when these types of errors occur. For views as a whole, their main objective is to handle POST and GET requests, so we will also be testing for malicious HTTP injections not already handled by Django. Overall, the views will mostly be handled by integration testing, but the unit tests we can run for the views will ensure that they are ready for integration testing, once we reach that stage.

# 3. Integration Testing

Integration testing is designed to verify that the separate modules, from CSV data ingestion and database queries to raster generation and web map creation, work together cohesively in a production-like environment. In this system, integration tests simulate the complete end-to-end process: executing the Django management commands that read in data from CSV to the database, generating the heatmap, converting the resulting GeoTIFF into a PNG, and finally creating the interactive Folium map with appropriate overlays and controls. These tests involve verifying that expected files, such as the heatmap raster and the final HTML map, are created in the correct directories that the Folium map includes essential elements like the ImageOverlay, LayerControl, and the custom BackButton when appropriate.

To simulate a real-world scenario, the integration testing suite begins by running the management command that aggregates data from multiple sources. This command retrieves grid coordinates from the database, parses CSV data for posterior median values, and processes this data to produce a GeoTIFF file. While unit tests ensure that each component functions correctly, integration tests validate the transition between each stage. For example, once the raster generation function completes, an integration test will verify that the GeoTIFF file is properly formatted and accessible for subsequent conversion steps. The conversion function then applies a custom blue-white-orange color map, using Matploblib’s LinearSegmentedColormap, and produces a PNG image, with tests confirming that the image file is saved correctly and that its pixel data reflects the expected smoothing and intensity adjustments.

Beyond file creation and conversion, integration tests further assess the web map creation process. After the PNG is generated, the system builds a Folium map by overlaying the image with defined geographical bounds derived from the raster. Tests verify that the map initializes with the correct center coordinates and zoom level, that the overlay bounds align accurately with the raster’s geographical extents, and that interactive elements such as the LayerControl and custom BackButton are present and behave as expected. In a simulated user journey, automated browser tools like Selenium can mimic user interactions, navigating to the map page, selecting filters, and clicking on navigation buttons to ensure that these actions trigger appropriate updates and overlays on the map. Automated browser testing plays a critical role in validating the overall user experience. By simulating real-world interactions, these tests ensure that when a user applies a bird species filter, the system correctly queries the database, updates the raster data, and refreshes the map display to show only the relevant markers. They also verify that navigation controls, such as the “View Larger Map” button and the custom back button, function seamlessly, directing users to the intended pages without errors. This comprehensive approach confirms that the data flows correctly through the pipeline and that the interactive components provide a fluid and intuitive experience.

Furthermore, integration testing includes stress testing and error handling scenarios. Tests simulate conditions such as missing or malformed CSV data, invalid database entries, and file I/O errors, ensuring that the system exits gracefully and provides meaningful error messages. If a CSV file is missing the required headers or if a database query returns no data, the integration tests verify that the management command logs the appropriate error and that the web map indicates the issues without causing a system crash. These tests help build resilience into the application, preparing it to handle unexpected real-world situations.

Overall, integration testing is critical for confirming that our mapping and visualization module integrates smoothly with the broader application infrastructure. By simulating full system workflows, from data ingestion through raster generation and ultimately to the interactive map display, these tests ensure that both the technical processes and user interface operate in harmony. This thorough, end-to-end verification not only confirms the correctness of the data flows between modules but also validates the overall user experience, paving the way for a robust and reliable deployment in a real-world environment.

# 

# 4. Usability/End-User Testing

Usability testing is the process of ensuring that the website/product functions as it is intended for the user. This way, problems can be prevented and resolved before establishing the product for the clients. The main goal is to solidify that the website is able to be navigated properly with little to no confusion. Another goal is for the user to gain an understanding of the background of the product and the information that it will display to the user, as the user navigates through the site. These goals are put into place to solidify that the product functions as intended, and the destination of the buttons are reached appropriately. The button saying map should not take the user to the home page but to the map page.

One such way of conducting a Usability Test would be to give a user, with no experience on working the site, some instructions to execute. These instructions can be either monitored in person or answered with a short questionnaire after completing the instructions. From these responses and noted reactions to the site, the team can make modifications to the product to better suit the user and ensure that the product is truly user friendly.

Some of the instructions will be as follows:

1. Read the main page content
2. Click on the button that says “Instructions”
3. Read the Instructions
4. Click on the button that says “Map”
5. View the map and interact with it
6. Select the bird species that you want to view
7. Update the page with these species by clicking “Update”
8. Once the new information is displayed, click on “View Larger Map”
9. Interact with the information and leave the map by clicking the back button
10. Extract the data by selecting “Export CSV”
11. Ensure the data has been properly exported and downloaded to your device
12. Navigate back to the home page by clicking the bird in the top right.

Some the questions that will be asked on the questionnaire will be as follows:

1. Did you find it easy to navigate throughout the site?
2. Were there any parts of the site that were causing confusion?
3. Would you recommend this site to people interested in learning about bird species impacts on environmental health?

These instructions and questions will help the team gain an understanding of what the general public thinks about the site. While the team is biased due to them creating the site, the general public can offer some insight and a new perspective into what should be added or modified in the site. The team will conduct an interview asking 1-2 people. During this interview, the team will ask the interviewee how they feel about the site and if the navigation through the site works as intended, solidifying that the site works as intended. In terms of the map, the interviewee will give feedback on if the buttons and filters for the map function properly and to ensure that they understand what is being displayed on the map.

The team will conduct these interviews shortly after submitting this document. From there, they will have a couple days to get their interviews done, that way they can meet on the following Wednesday to discuss the findings. It is from these findings that a new plan will be drafted to focus on the aspects that the interviewees deemed necessary for the site. These new implementations will be applied to the system, and new interviews will need to be conducted. The same interviewees can participate and ensure that any requests or recommendations suggested were achieved or if there is more that needs to be done on the site.

# 5. Conclusion

Several forms of testing are required to verify the functionality of the project. This ranges from the project as a whole, down to the smallest function written in the project. Starting off the process, unit tests will be conducted to check that each function is able to run properly. Upon this success, integration tests will follow. They will help to verify that the data processed from the front end is properly communicated to the backend, where it will generate the appropriate data and send information to the front end. From this, the information should be capable of displaying the appropriate information from communication with the backend. Once both tests are completed, the team will ask people to participate in a questionnaire that tests the navigation and the functionality of the site. From these responses, new ideas for the project can be generated, or improvements can be made for a better user experience. This process may seem long, but it is a necessary step in helping the team create a useful tool for our clients and the public.