Software Testing Plan

4/4/2025

Forest Frames & Data Integrity



Sponsored by Dr. Camille Gaillard, Dr. Chris Doughty, Dr. Duan Biggs, and Dr. Jenna Keany

Mentor: Scott Larocca

Forest Frames Members: Dalton Tippings, Daniel Austin, and Nicholas Greco

Data Integrity Members: Aidan Trujillo, Fady Zaky, and Kyle Bambling

Version 1.0

# Table of Contents

# 

[**Table of Contents 2**](#_wti665xp3fed)

[**1. Introduction 3**](#_txr0hxfzj5p2)

[**2. Unit Testing 3**](#_gb39gqtid1nz)

[2.1 App 4](#_jctxduhoxoop)

[2.2 Server 6](#_ag7tw2x8g1k3)

[2.3 Database 8](#_ae4vxwbpyvt8)

[**3. Integration Testing 11**](#_vc15s9n3hege)

[3.1 App & Server 12](#_oo7vvbq6t59s)

[3.2 Server & Database 13](#_seluri5lenwz)

[**4. Usability (End-User) Testing 15**](#_st7trn7f81p1)

[4.1 App Usage 15](#_iuw8df8ljvo7)

[4.2 Backend Interaction 15](#_gr36r9pfzjr8)

[**5. Conclusion 16**](#_46k9ryu0tsdt)

# 

# 1. Introduction

In recent years, citizen science has emerged as a transformative approach to environmental conservation by utilizing crowdsourced data to support wildlife monitoring, biodiversity research, and ecological health assessments. This is especially valuable in regions with large, difficult-to-monitor wilderness areas such as Malaysia and Kenya. To address the challenges of limited resources and inconsistent ecological data collection in these rural regions, our senior capstone project introduces a citizen science mobile application designed to incentivize local residents to participate in environmental data gathering.

The app allows users to locate survey points requested by scientists, navigate to them using offline maps, and submit flora and fauna observations that are automatically verified through GPS tracking, machine learning, and integration with existing wildlife databases. Users receive rewards for submitting valid data, creating a sustainable loop of engagement and contribution to conservation efforts.

As this project spans multiple components, mobile frontend, server backend, and a MySQL-based database, ensuring the robustness and correctness of each module is critical. To support this, we are implementing a comprehensive testing suite focused on unit testing and integration testing. These testing practices will help us verify that individual components behave correctly on their own and function cohesively when interacting together.

Unit testing will validate the behavior of specific functions and modules in isolation, enabling quick feedback and reducing the risk of regressions. Integration testing will ensure that interactions between the application, backend server, and database produce the expected results under various real-world scenarios.

By selecting the appropriate testing libraries for each component, we can build a reliable and maintainable application that is both user-friendly and scientifically trustworthy.

# 2. Unit Testing

Unit testing is a type of testing that is meant to ensure that the small parts of your code, such as functions, work as intended. This is usually done through tests called assertions that take input from the code you are testing and compare it to a preset value. If the two values match then the test passes. This is especially useful when the results for a piece of code need to be tested repeatedly when changes are made. These tests ensure that the logic for the functions works as expected and there are no unknown bugs present.

Unit Testing Tools Summary:

1. App: The primary testing tools and frameworks used for the user-facing application will be JUnit. This library provides a wide array of tools that can be used to conduct a wide variety of tests. Additionally, it integrates well with Kotlin so that we are able to test within Android Studio while developing our code. JUnit provides the abilities to test all parts of our application including the res files and the actual code itself.
2. Server: Since most of the server code is written in Python, the best choice for unit testing would be Python’s own library specifically built for it, “unittesting”.
3. Database: The database operations file is written in Python, the libraries “pytest” and “unittesting” are great for testing databases written in mysql.

## 2.1 App

For android development, there are a wide variety of tools and services offered that can be used to test the consistency and functionality of the application. The primary library for testing in android app development is called JUnit. JUnit provides a variety of tools for testing many aspects of android development and is the standard for many applications and testing usages.

The purpose of unit tests both during development and usage is to make sure that the application is acting in an expected manner. For unit testing this involves ensuring that logic is consistent and acts as intended during the app's use. So for this application unit testing will primarily focus on ensuring that individual pieces of code do what is expected, such as saving information, manipulating data, and performing expected operations on the phone. Testing also involves ensuring that the application is running efficiently which involves the usage of the android profiler to test what pieces of logic use up the most resources and time.

JUnit is a very modular testing program and it will allow us to test logic in the order that we need. For our particular application, this is important because we need to verify the integrity of the data that we collect. We need to ensure that the functionality of the verification tests is working properly. This will save us time and ensure that what we have is working properly without having to run through those trivial tests each time, such as making sure a function that checks a user location when given their coordinates, returns the correct result.

### 2.1.1 Location Authentication

Test whether the app properly authenticates the location of the user when a picture is taken.

**MODULE DESCRIPTION** - The function verifyDistanceToCoord() determines the user's current distance from a selected coordinate. This can then be compared to the GEDI coordinate to determine if the user is within the coordinate.

**TEST DESCRIPTION** - Junit will use assertions to determine the accuracy of the function and comparisons. First will be determining if the location of the provided coordinates is accurate by setting mock coordinates for the application and determining if the function is accurate. Then it will be verifying that the provided coordinates are within the area around the GEDI coordinates.

**CONSIDERATIONS** - This will be tested under the cases of, access and no access to the internet, varying degrees of GPS precision.

### 2.1.2 Proper Image Saving

Test whether an image is successfully saved to local phone storage.

**MODULE DESCRIPTION** - The function takePhoto() is called when a user tries to take a photo, upon the photo being taken it should save it to the phone's internal memory.

**TEST DESCRIPTION** - This will be tested through saving files both successfully and unsuccessfully with preset file names to internal storage and being able to detect those files in that storage if they exist.

**CONSIDERATIONS** - This will be tested based on varying phone specifications, such as storage availability, resource limitations, and file system setup.

### 2.1.3 Proper image retrieval from internal storage

Test whether the app is able to properly read and retrieve images stored on the user's device.

**MODULE DESCRIPTION** - The two functions createImageUploadList() and createAudioUploadList() both take the user-recorded data, lists it to the user, and prepares the data for upload to the server.

**TEST DESCRIPTION** - This will be tested through having preset sets of user data and testing if the audio and image lists match the data that is saved into the phone.

**CONSIDERATIONS** - This will be done on varying amounts of data and file types.

### 2.1.4 Proper retrieval of nearby coordinates

Test whether the app is able to retrieve coordinates near to a given user location.

**MODULE DESCRIPTION** - The function displayGEDICoordinates() takes in a given long and lat coordinates that are meant to simulate user coordinates.

**TEST DESCRIPTION** - This will be tested by inputting user coordinates located in crowded GEDI areas, areas with low number of GEDI footprints, and areas with no GEDI coordinates (usually over water).

**CONSIDERATIONS** - This will be tested on different user locations around the world.

### 2.1.5 The distance between two coordinates

Test whether the app is able to calculate the proper distance between two given coordinates.

**MODULE DESCRIPTION** - The function calculateDistance() will take two sets of coordinates as input to the function and then return a distance in km.

**TEST DESCRIPTION** - This will be tested by subbing in coordinates with predefined distances from each other. It will verify that the function does indeed return the expected distance within a certain error (since these are stored as doubles).

**CONSIDERATIONS** - This function is used in determining if the user is within a certain radius, so it is very important to the integrity of our data. It should be able to account for very small distances (less than 5m).

## 2.2 Server

### 2.2.1 Request: Upload Data

Test that the upload data request handler returns proper responses based on given request data.

**MODULE DESCRIPTION** - The upload data request handler accepts incoming requests from the app which carry the data users have collected from the app, along with the corresponding metadata. The request handler verifies the authenticity of the user, the information sent over, and makes a call to store the retrieved data in the database.

**TEST DESCRIPTION** - Using Python’s “unittesting” library, a mock request will be created with proper/improper data, and the handler’s return will be examined to determine the test’s success/failure.

**CONSIDERATIONS** - The following will be accounted for in the test: empty request, request with invalid user info, request with improperly formatted metadata, and a valid request.

### 2.2.2 Request: Retrieve Verification Status

Test that the retrieve verification status handler returns proper responses based on given request data.

**MODULE DESCRIPTION** - The retrieve verification status request handler accepts incoming requests from the app which contains information about the user. This information is then used to acquire the verification statuses of data which the user has uploaded to and through our server, and returned to the user’s app to be displayed.

**TEST DESCRIPTION** - Using Python’s “unittesting” library, a mock request will be created with proper/improper data, and the handler’s return will be examined to determine the test’s success/failure.

**CONSIDERATIONS** - The following will be accounted for in the test: empty request, request with invalid user info, a valid request, and that the correct amount of statuses are returned.

### 2.2.3 Image Verification For Animals

Test that the image verification module for animals conducts proper operations based on given image data.

**MODULE DESCRIPTION** - The animal image verification module accepts the name of an image file stored on the current system. The image is then run through a detection/classification process by the Python library “PytorchWildlife”, the relevant information is acquired, and then a verification status is stored.

**TEST DESCRIPTION** - Using Python’s “unittesting” library and test images acquired from sponsors for this project, an image will be passed into the module and the status it sets will be examined to determine the test’s success/failure.

**CONSIDERATIONS** - The following will be accounted for in the test: no image given, poor quality image, image with no animal, and multiple images with animals (to help adjust threshold values specified in prior documents).

### 2.2.4 Coordinate Filtering

Test that the coordinate filtering operation filters a given coordinate list based on certain specifications.

**MODULE DESCRIPTION** - This module is a function that accepts a list of coordinates, a specific coordinate to filter around, and a limit for the amount of coordinates in the filtered list. The function then creates a KD-Tree to efficiently determine the coordinates closest to the designated one, and returns that new list.

**TEST DESCRIPTION** - Using Python’s “unittesting” library, sets of created coordinate lists (or possibly ones queried from our database), and designated values for specific coordinates and filter limits will be passed into the function. The function’s returned coordinate list and the time it took to perform its operations will be examined to determine a test’s success/failure.

**CONSIDERATIONS** - The following will be accounted for in the test: no coordinates or other empty/null arguments given, different-sized coordinate lists, sparse coordinate list, and large coordinate lists from our database.

## 2.3 Database

### 2.3.1 Database Connection Handling

Ensure the database connection is successfully initialized or appropriate errors are raised when credentials are incorrect.

**MODULE DESCRIPTION** – The connect() method initializes the connection to a MySQL database using environment variables. This is triggered in the constructor of DatabaseManager.

**TEST DESCRIPTION** – Use mocking to simulate both successful and failed connection attempts. Python’s unittest.mock will mock mysql.connector.connect to test different error conditions and validate exception handling.

**CONSIDERATIONS** – Tests include valid credentials, missing environment variables, and incorrect credentials. Also tests if self.connection is correctly set or remains None upon failure.

### 2.3.2 Table Reset Function

Verify that reset\_tables() clears relevant database tables and resets auto-increment.

**MODULE DESCRIPTION** – reset\_tables() disables foreign key checks, truncates users and photo\_metadata tables, and re-enables the checks. It uses direct SQL execution with commit control.

**TEST DESCRIPTION** – Use a test database with mock data. After calling reset\_tables(), verify that both tables are empty and have reset their auto-increment values. Use unittest + test fixtures.

**CONSIDERATIONS** – Includes cases where the table is already empty, when foreign key checks are already disabled, and when the connection or cursor fails mid-operation.

### 2.3.3 User and Asset Deletion

Validate that a user and their related assets are correctly deleted from both S3 and the database.

**MODULE DESCRIPTION** – delete\_user\_and\_assets() deletes user data and their image references in RDS, and also removes related image files from the S3 bucket.

**TEST DESCRIPTION** – Use mocks for both S3 (boto3.client) and MySQL cursor. Insert test user and image data, then check that corresponding S3 deletions and DB deletions are triggered and complete successfully.

**CONSIDERATIONS** – Tests successful deletions, S3 deletion failures (e.g., network errors), nonexistent users, and DB rollback correctness on failure.

### 2.3.4 Soft Delete Image

Confirm that soft\_delete\_image() marks an image as deleted in the database and moves the image to an S3 "trash" folder.

**MODULE DESCRIPTION** – This method finds the image file path in the database, copies it to an S3 trash/ location, deletes the original, and updates the record with a deletion timestamp.

**TEST DESCRIPTION** – Mock database results and S3 actions. Assert that copy\_object, delete\_object, and the SQL UPDATE query are called as expected. Validate correct S3 key generation.

**CONSIDERATIONS** – Handles nonexistent images, database failure after S3 move, and S3 copy/delete failure mid-process.

### 2.3.5 Hard Delete Image

Test that delete\_image() removes both metadata and file completely.

**MODULE DESCRIPTION** – This method deletes an image object from S3 and removes its metadata from the database, but only if the image belongs to the specified user.

**TEST DESCRIPTION** – Simulate user-image mappings and verify only rightful deletions occur. Use mocks to simulate various failures (missing image, S3 delete error, etc.).

**CONSIDERATIONS** – Includes boundary case for non-matching user/image pairs, nonexistent records, and S3 permissions issues.

### 2.3.6 GEDI CSV Coordinate Import

Test import of coordinate data from a CSV file.

**MODULE DESCRIPTION** – import\_gedi\_coordinates() opens a CSV file, parses rows, and inserts latitude/longitude into a MySQL table, using Python’s Decimal for precision.

**TEST DESCRIPTION** – Use temporary CSV files (via Python tempfile) with different row formats. Mock database cursor to track insertion calls and catch exceptions.

**CONSIDERATIONS** – Tests malformed CSV (missing lat/lon), empty CSV, extra columns, file-not-found, and database insertion errors.

### 2.3.7 Coordinate Fetching

Verify that fetch\_coordinates() returns a limited list of coordinates from the database.

**MODULE DESCRIPTION** – Fetches the top 200 rows of gedi\_coordinates as a dictionary list using a simple SQL query.

**TEST DESCRIPTION** – Mock cursor and return values. Validate output structure, row limits, and behavior on connection/cursor failure.

**CONSIDERATIONS** – Includes empty table return, data type formatting, and cursor exceptions.

### 2.3.8 Image Storage

Verify images are uploaded to S3 and that the metadata is stored properly in RDS.

**MODULE DESCRIPTION** – store\_image() uploads an image file from the server to the S3 bucket and logs its meta data in the photo\_metadata table in the RDS.

**TEST DESCRIPTION** – Use local test images to imitate database insertion. Afterwards verify S3 storage and file metadata. Rollback insertion on error.

**CONSIDERATIONS** – Includes missing local file, large file, failed upload, and DB insert failure. Verifies consistency between S3 and DB states.

### 2.3.9 Image Retrieval

Test that fetch\_image() pulls the correct file from S3 and stores it locally.

**MODULE DESCRIPTION** – Uses download\_file from S3 and reconstructs the local filename and directory. Verifies DB entry before download.

**TEST DESCRIPTION** – Use mocks to simulate correct and incorrect file paths, user ownership mismatch, and S3 download errors.

**CONSIDERATIONS** – Includes test cases for nonexistent image ID, permission denial, and file overwrite prevention.

### 2.3.10 User Creation

Check that create\_user() correctly inserts user credentials.

**MODULE DESCRIPTION** – Simple insert query to add username and hashed password into the users table.

**TEST DESCRIPTION** – Mock cursor execution. Validate that usernames are unique and that commits happen as expected. Add a negative test for duplicate users.

**CONSIDERATIONS** – Tests SQL injection edge cases, blank usernames, and rollback handling.

# 3. Integration Testing

Integration testing is the process of testing the interactions between distinct modules of a system. This involves the use of a test harness to track how each module interacts with each other. The goal of this kind of testing is to ensure that each module properly interacts with the other modules. For example, ensuring that when the user facing app interacts with the server, it is both sending and receiving information in a proper and intended way. Along with this ensuring that the server is properly storing information in the database and is able to properly get data from the database.

We decided on three different larger-scoped modules for our integration test: the app, our server, and our database. All of these modules have specific entry/exit points for request and queries, so it will be imperative to ensure the transition from one module to another is thoroughly tested. Though we have three modules, integration testing from the app to the database is not needed, as the app will only have indirect access to the database via the server passing their information, thus the main two types of integration testing will be conducted are the app-to-server and server-to-database connections.

## 3.1 App & Server

The application only needs to connect with one external module and that is the server since the server will facilitate any other connections that need to be made. The app will need to connect with the server for the retrieval of coordinates, the sending of data, and the retrieval of the verification. Additionally the user data will need to be transported to and from the server when an account is created and when an account is signed in with. To simulate these requests, we will utilize MockWebServer within Android Studio so that we can simulate the HTTP requests that our Python server will use. For every request we need to have going to the server, we will have an HTTP request that is made to send it to the server and begin that process on the server. This is especially helpful so we don’t have to have a constant query to the actual server when we are just trying to test. This also integrates well with kotlin and makes it a perfect choice for our application.

The module integration is verified because we can ensure that any response we will be ready for. This includes when we cannot connect for example if the user is not connected to the internet. We can ensure that the app properly handles data when cases like that happen so nothing is deleted. The unit testing done on both the server and the application itself should cover all of the bases for what happens on each, so it won’t be necessary to have the app connect to the server for this since we know the server will work. So having a mock server will be sufficient for us ensuring that it handles data from the server as expected.

### 3.1.1 Uploading and Fetching User Data

Ensure that the app’s upload data and data retrieval operations work correctly.

**MODULE DESCRIPTION** – Both functionality includes uploading image/audio data to the server and retrieving verifications statuses of that data, both via a POST request.

**TEST DESCRIPTION** – Make a POST request to the mock server to upload an image and receive back that it was stored. Another POST request will then be made to retrieve the status of the image that was sent in the prior request.

**CONSIDERATIONS** – Tests that the POST requests are properly formatted to test with the mock server and vice versa, along with testing the app properly formats the data and status returns it gets back.

### 3.1.2 Coordinate Retrieval

Ensure that displayGEDICoords() works as expected.

**MODULE DESCRIPTION** – A function that makes a server request for nearby GEDI Coordinates based on a desired location that are then displayed on a map in this function.

**TEST DESCRIPTION** – Make a POST request to the mock server sending a desired center coordinate and for x amount. The mock server will send back a list of x coordinates that should just be in a cluster around the center point.

**CONSIDERATIONS** – Tests that the function can properly handle a response from the server and tests that it makes a proper request with the correct information and formats it correctly before sending to the server.

### 3.1.3 Sending Data

Ensure that uploadDataToServer() works as expected.

**MODULE DESCRIPTION** – A function that sends the data collected to the server to get verified.

**TEST DESCRIPTION** – Make a POST request to the server sending the files of either audio or picture. Receive different HTTP responses and handle them.

**CONSIDERATIONS** – Tests that the app will not break and data will not be lost if the requests are unsuccessful. Tests that when an OK request is received, the app will make sure the photos are handled properly on the device.

## 3.2 Server & Database

Since the server acts as a middle-man between the app and the information stored in the database, integration testing must be conducted between the server and the database. We will be using GitHub Actions services to conduct testing between the server and our actual database, along with testing between the server and a mock database.

We can verify this module integration because we can ensure that each specific query and its result from the database, those which are integral to the full system’s functionality, are tested both in this integration testing and in the aforementioned unit tests. These tests will include edge cases which could have resulted in an incomplete or improper operation such as queries with empty or invalid parameters.

### 3.2.1 User Authentication

**MODULE DESCRIPTION** – This is a function that queries the database to check if a user’s login information (username and password) are in there. Used for both logging a user in or creating their account info in the database.

**TEST DESCRIPTION** – The test will involve passing in user information as parameters, and using either a mock database or actual, and the query’s return will be examined to determine test success.

**CONSIDERATIONS** – The following will be accounted for in the test: no user information given, invalid user info (user not in database), and valid user info.

### 3.2.2 Storing User-Collected Data

**MODULE DESCRIPTION** – This is a function which takes user-collected data (images/audio) and their corresponding metadata, and stores it within the database under their information.

**TEST DESCRIPTION** – The test will involve passing in user information and the user-collected data as parameters, and using either a mock database or actual, and the query’s return will be examined to determine test success.

**CONSIDERATIONS** – The following will be accounted for in the test: no information given, invalid user info, invalid file type being stored, and valid data being provided.

### 3.2.3 Retrieving Coordinates

**MODULE DESCRIPTION** – This is a function which retrieves a list of coordinates from the database, which would then be filtered before being sent back to the app.

**TEST DESCRIPTION** – The test will be using either a mock database with a manually-inputted list of coordinates or our actual database, and the query’s return will be examined to determine test success.

**CONSIDERATIONS** – The following will be accounted for in the test: no coordinates stored in database, large amount of coordinates stored (list that exceeds 200 query limit), and medium amount of coordinates stored (less than 200 limit).

# 4. Usability (End-User) Testing

Usability testing is the process of testing the system with end users who will be using the system. This is done after the initial functionality of the system is created and the previous tests have been implemented. Usability testing is used to test whether the implementation is usable by users and fulfills the requirements that said end users want. This is to ensure that the features that have been implemented are actually what is required for the application and to get feedback from users to make any changes that would make the system more suitable for their needs.

## 4.1 App Usage

### 4.1.1 Unguided Ease of Use

Trial users will be given the app and will attempt to use its functionalities without developer guidance or input.

**BACKGROUND** - Due to the target audience of this app being those who do not use technology significantly, it is important to determine how usable the application will be for those who will use the app without prior knowledge of its functionality.

**DESCRIPTION** - This test will be conducted by taking a sample group of people who have not been exposed to our app and asking them to use it. We will record what functionality they are able to use and how quickly they can use it. Following this the test users will provide feedback on the app, what was clear and what was confusing, along with what they think can be improved. This feedback will then be incorporated into future development decisions.

## 4.2 Backend Interaction

### 4.2.1 Interpreting Server Logs

Individuals will be shown server logs containing information on app/server interaction and will determine if that output is human-readable.

**BACKGROUND** - Since this project will eventually be passed to our sponsors, and in the future potentially another software development team, it is crucial that the messages produced by the server while it is in use are readable and not confusing enough to help maintainers analyze the system or debug issues.

**DESCRIPTION** - This test will be conducted by giving members of a test group, which would potentially include our sponsors, a comprehensive text output of our server log. They would then be asked to describe what certain lines of output mean or be asked to search for an output line that matches a description we give them. We will record how accurate their responses are and calculate an overall percentage based on their accuracy for each question. This overall percentage will inform us on how easily others are able to read that output and what lines of output need to be modified for increased readability.

# 5. Conclusion

The three main components of this project are the app, server, and database. We are unit testing the main elements of each of these components to verify that they work properly. In the app, this is using JUnit to test that location and image handling are functioning correctly. It is crucial that the app can accurately authenticate the user’s location in the app. The server’s main functions and what we are testing is data upload, user information verification, and image verification.

One of the most important parts of this project is verifying that images uploaded have not been tampered with. This along with ensuring that the images can be uploaded are important to the project and the server’s functionality. The database is used to store user and data information. The testing done on the database is mainly to ensure that data can be deleted to make room for new data, user creation, image handling, and connecting to the database. These are done to make sure that the data gathered and verified by the app and server can actually be stored and used. Each of the three components in the project are all important but we also need to test the project as a whole. Integration testing is done to verify that the different components work together correctly. The app and server communicate to upload and retrieve data. If this failed, the user would not be able to see needed information like GEDI locations and they may need to regather or reupload data if it fails to upload. The server and database communicate primarily to move and store authenticated data.

Testing will be done to ensure that storing data and GEDI coordinates along with sorting the GEDI coordinates will work as expected. The least critical tests to be done is usability testing. This is important to make sure that users are able to easily interact with the app. Although the project can work without this testing, we still need to make sure that the users can navigate the app and use it properly. App testing is done to verify that location and image handling works. Testing on the server guarantees that information is properly uploaded and authenticated. Database testing confirms that stored data can be modified and usability testing ensures that users can deftly navigate and use the product. All of this testing is done to ensure that we have a functional, easily accessible app that correctly authenticates user gathered data.