WSUDAS-AI

Prototype Project Report - Draft

WSUDAS

WSUDAS-KS

Son Nguyen and Kieran McHugh

12/1/2025

TABLE OF CONTENTS

I. Introduction

Agricultural institutions and farmers are responsible for producing the food we eat every day, and the agricultural industry is a major component of the U.S. economy, accounting for over a trillion dollars of the United States' G.D.P. [1]. Because it is such a large and valuable industry, development and utilization of more efficient practices can save farmers (and in turn consumers) hundreds of millions of dollars.

Washington State University Decision Aid System [2] is a service designed to help farmers and other agricultural institutions predict various factors that affect their crops, ranging from weather, what insects are present, what the populations of those insects are, and when pesticides should be applied to protect crops. Large quantities of data are collected from a multitude of sensors and processed by DAS. Users can then choose nodes and use the data collected at those locations to better inform their decisions and practices.

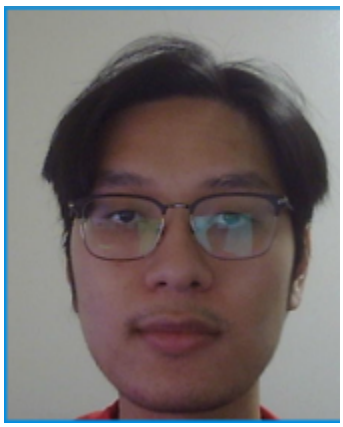
While DAS has proven very effective in aiding farmers, it was originally tailor-made specifically for informing farmers growing potatoes, but its scope could be expanded much further than that. The data being collected could be applied to a multitude of other crops were the system to be expanded, and that is now what WSUDAS seeks to do.

II. Team Members

II.1. Kieran McHugh

Kieran McHugh is a senior majoring in computer science at Washington State University. His interests include machine learning and artificial intelligence as well as data science. Kieran has worked with Python, R, C/C++, and HTML, and has worked with the Amazon Web Services platform. For this project he has primarily focused on writing the Scoutlabs data transfer aspects and scheduling the transfer job.

II.2. Son Nguyen



Son Nguyen is also a Senior at Washington State University, majoring in computer science. His coursework shows a slight interest in data science, machine learning related topics, and a slight interest in cyber security. Son has some experience in Java, Python, C#, and SQL queries. For this project, he works mostly on similar topics to Kieran.

III. System Requirements Specification

III.1. Use Cases

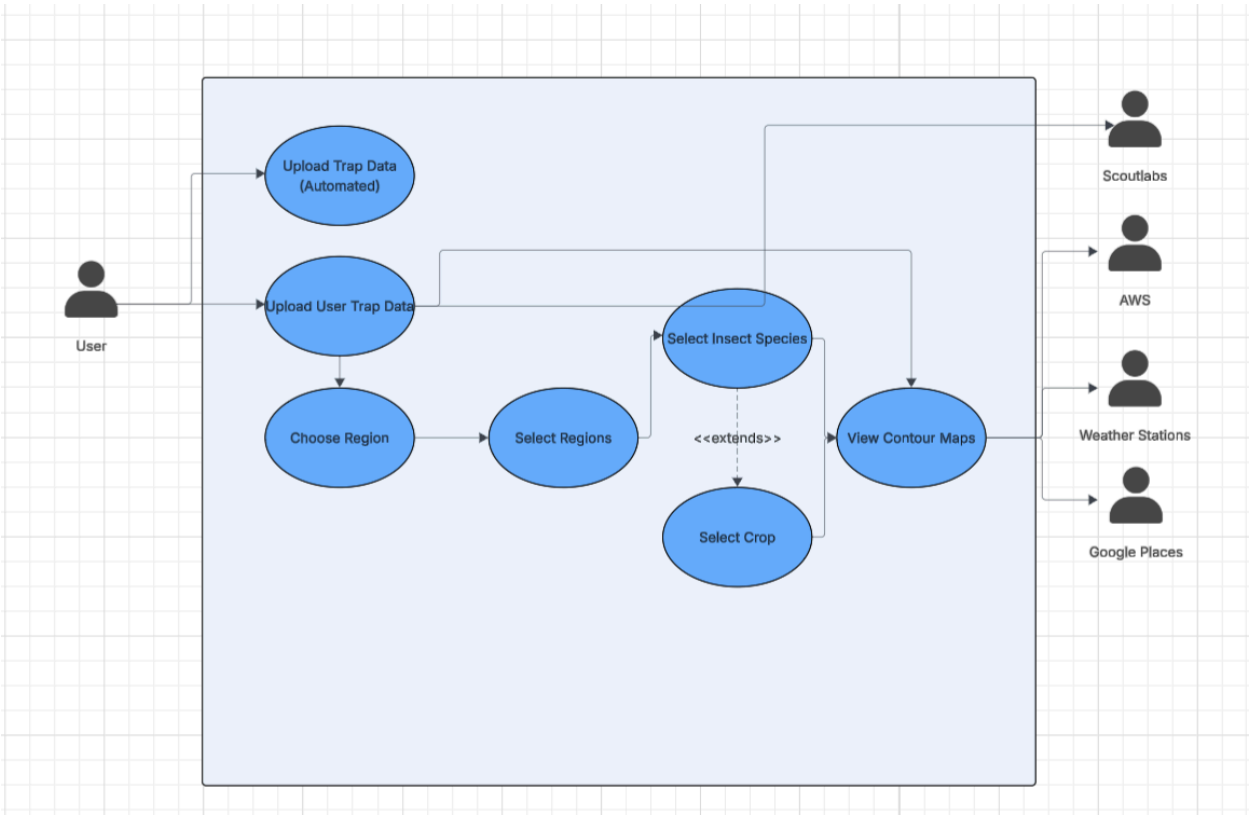


Figure 1: Use Case Diagram

Figure 1: Describes how a typical user would interact with the WSUDAS website

- UC-1: Trap data is uploaded to WSUDAS
    - Actors: Customer/User with Scoutlabs Trap
    - Precondition: Trap is connected to WSUDAS' database
    - Postcondition: Trap data will automatically appear in WSUDAS' database
    - Basic Path:
        - User sets up their Scoutlabs trap and connects it to WSUDAS
        - When the trap is ready to upload data, the data is automatically pushed to WSUDAS
        - Data is cleaned and entered into the model/contour system
    - Alternative Path:
        - (Also the current system) User uploads .csv file manually to WSUDAS' database
        - Data is automatically cleaned.
    - Related Requirements:
        - Connection to WSUDAS
        - Scoutlabs API integration

- UC-2: Scoutlabs Data Included in Contours

- Actors: User/Developer
- Precondition: Scoutlabs traps have uploaded their data to WSUDAS (either manually or automatically)
- Postcondition: Insect data from the Scoutlabs traps is used when generating contours
- Basic Path:
  - User logins into the DAS website
  - Scoutlabs trap data has been sent to DAS (either by them or other users)
  - User navigates to the contour maps and selects an insect
  - Insect data and contours displayed to the user include the data uploaded by the Scoutlabs traps
- Related Requirements
  - The user (if not a developer) has subscribed to WSUDAS
  - Scoutlabs traps have uploaded insect data

III.2. Selected User Stories

- US-1: Trap Data Upload
  - As a user, I would like to have my data automatically uploaded from my Scoutlabs traps to WSUDAS' database.
  - Feature: Trap Data Upload
  - Scenario: Trap Uploads Data
    - Given the user has a Scoutlabs Trap
    - When the trap collects insect population data (either daily, weekly, or monthly)
    - And the data is ready to be exported
    - Then the Scoutlabs trap will automatically send the insect data to DAS
- US-2: Scoutlabs Data Included in Contours
  - As a user or developer, I would like to see the insect data collected from Scoutlabs traps used when generating contours on the DAS website.
  - Feature: Scoutlabs Data Included in Contours
  - Scenario: Data is used in contours
    - Given Scoutlabs traps have uploaded data (either manually or automatically)
    - When the data is entered into WSUDAS' database
    - And the user selects an insect's contour map to view on DAS' website

- Then the displayed contour map will reflect and include the data uploaded by the Scoutlabs traps.

III.3. Traceability Matrix

Functional Requirement	Use Cases	User Stories	Priority
Data Upload	UC-1	US-1	Level 0
Data Storage	UC-1	US-1	Level 0
Data Integration	UC-1, UC-2	US-1, US-2	Level 1
Contours	UC-1, UC-2	US-1, US-2	Level 2

III.4. Functional Requirements

ID	Requirement Name	Description	Source	Priority Level
FR-01	Automated Data Ingestion	The system must be able to collect data automatically from the smart traps with minimal human intervention	The End User	0
FR-02	Smart Traps Integration	Going from FR-01, the system must be integrated with smart traps APIs, namely Scoutlabs, to automatically collect and digest data for future usage. It should be able to throw errors and notify the developers if something goes wrong	WSUDAS Team	1
FR-03	Contour Map Generation	The System must be able to generate contour maps based on the data provided by the traps. If trap data is missing, it must still be able to generate maps based on available data	WSUDAS Team	0

FR-04	Unified Spatial-Temporal Model	The system must unify both the time and space models into one model to provide more precise suggestion	WSUDAS Team	2
FR-05	Insect Abundance on Map	The system must render insect abundance on existing map	The End User	1
FR-06	Pin Interaction	User can interact with map pins for localized data and basic suggestion before going into detail	The End User	2
FR-07	Insecticide/Ovicide Timing	The system must provide suggestion or guidance in an advanced manner so that user can plan accordingly	The End User	0

III.5. Non-Functional Requirements

NFR-01: The data collection automation part must be simple enough for normal users.
NFR-02:The models should be functional on more than 1 treefruits.
NFR-03: The system should be scaled to work on at least 3 different locations without interfering with one another.
NFR-04: The suggestion provided by the system should be at least 1 week earlier for better planning.
NFR-05: The system should be able to update information daily for better accuracy.

IV. System Evolution

We do not anticipate the integration of traps to cause too many issues for us over the course of the project as APIs are provided. However, we were recently made aware that our client is looking to move away from using PHP and is currently working with another capstone group to accomplish this, likely rewriting much of their existing codebase in the process. However, our client did not make us aware of this nor mention it as something they want us to account for, and have in fact told us to write what we do in PHP, so this is not a major priority for our group in particular. Still, moving forward we will try to take this knowledge into consideration and ensure that whatever we make for our portion of the project will still be functional under a new framework.

V. Architecture Design

V.1. Overview

For the structure of our project we are somewhat limited by the constraints of the existing system. Making even seemingly small changes to the overall design schema would require refactoring enormous portions of the existing project and is therefore not desired by the client.

As such, any design decisions we make must also adhere to the current structure of WSUDAS' services and cannot conflict with or change them.

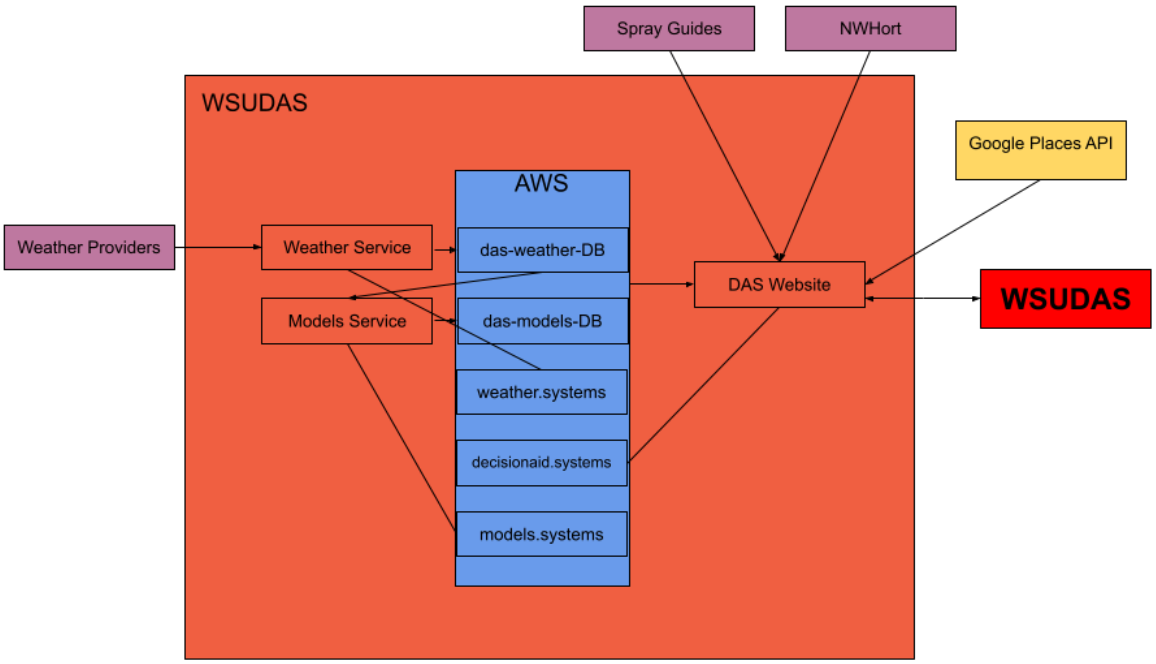


Figure 2: Block Diagram of DAS System

In the above model, **Weather Providers** give the basis for data used in WSUDAS' system. This data is then fed into the **Weather Service** component which then transfers the data to the **Amazon Web Services databases** where this data is stored. The **Models Service** then utilizes this data to generate the models the user sees. These models are then transferred to another database (**Models DB**) and are then propagated to the **DAS Website**. Utilizing some additional third party software, these models can provide users with information on when to make sprays as well as show a map of the region (using **Google Places**), and this map and models then show the generated contours to the user.

V.2. Subsystem Decomposition

Since this is already an existing system, it can be broken down into three main subsystems: the weather services, the model services, and the website.

III.2.1 Weather Service

a) Description: Ingests weather observation from appointed sources, validate and pre-processing the data
b) Concept and Algorithm Generated:
    i) ELT and possible normalization: data provided would be cleaned, transformed, and normalized before getting saved into the database
    ii) Schema design: the schema design and mapping of data collected

c) Interface description:
    i) Service required: Data Storage (AWS), Weather station for data collection, Database (MySQL)
    ii) Service provided:
        1) Cleaned Data to Model Service for further analytics. The input would be raw data from stations and traps. The output is processed data.
        2) Data to Web Service for displaying basic information. The input is cleaned data from the first step, the output is data successfully loaded into database for future usage

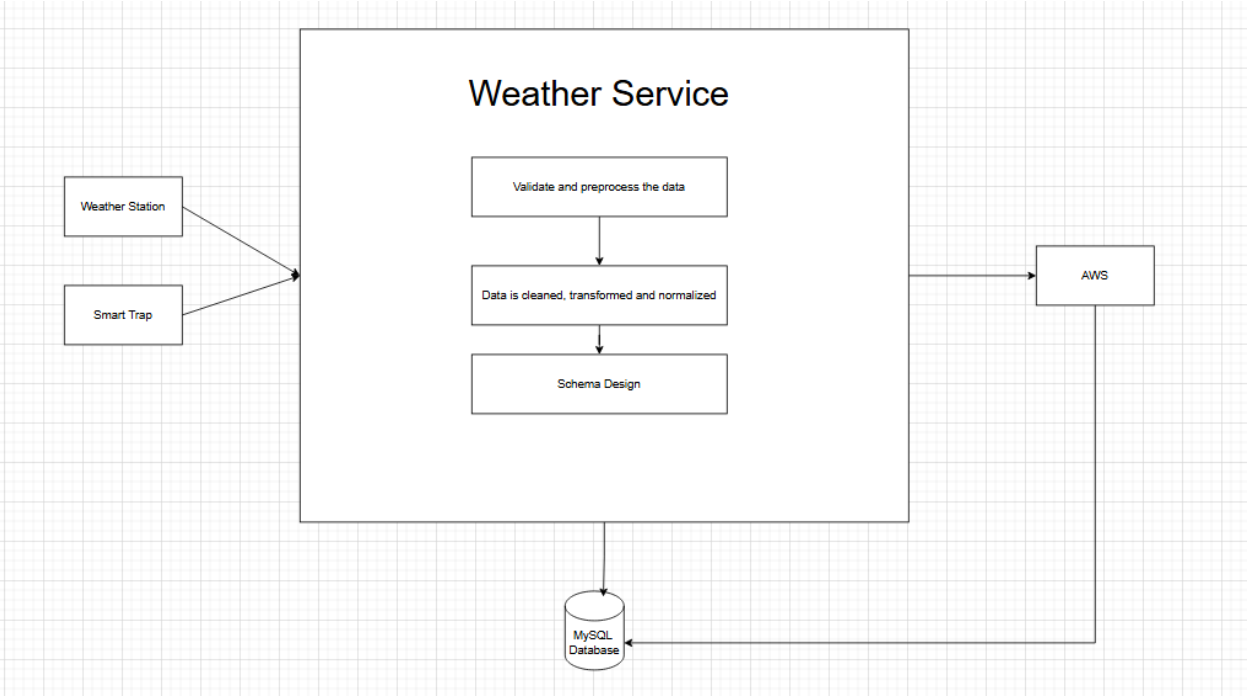


Figure 3: Weather Service Visualization

III.2.2 Model Service

a) Description: Run models based on the data provided by the Weather Service, produces contour features, writes outputs to store in the database and publish it to the website
b) Concept and Algorithm Generated:
    i) Model design: Design decision models for spraying, insects development, and so on
    ii) Spatial interpolation: configure contours for regions with preset tailored to them
c) Interface description
    i) Service required: Data Storage (AWS), Database(MySQL), Weather Service for data, Google Place
    ii) Service provided:

1) Contour maps in GeoJSON to Website for showing the location applied to the suggestion. The input is cleaned data from the database, the output is GeoJSON file helping generating the contour maps
2) Suggestion to Website for giving recommendation for users to follow. The input is the cleaned data and the GeoJSON file, the output is suggestion based on the cleaned data from the past til present and the location

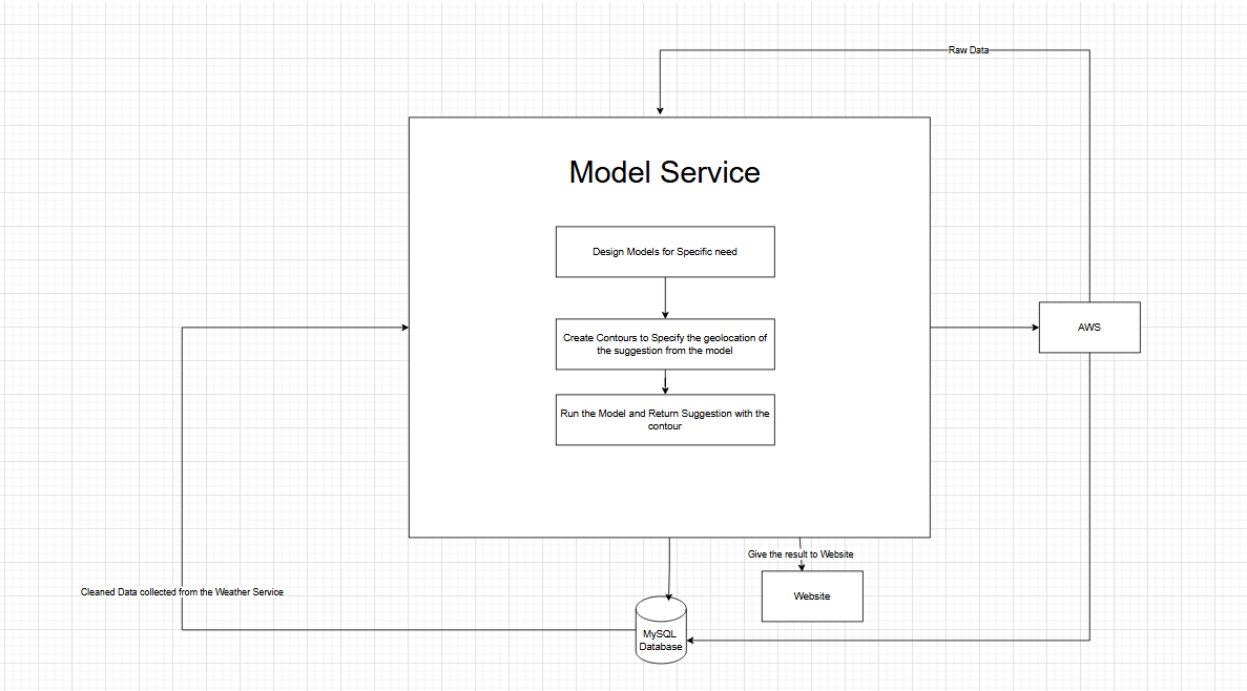


Figure 4: Model Service Visualization

III.2.3 Website

a) Description: User-facing website that visualize the models output as contour maps, show station information, and provide suggestion
b) Concepts and Algorithms Generated:
    i) Map UX: Provide a map based on Google Places/Maps, layer toggle, time sliders
    ii) Accessibility: color scales for contours, basic data visualization
c) Interface description:
    i) Service required: Weather Service for station basic data, Model Service for contours and suggestion, Authentication for login, Google Places/Maps for map and geocoding
    ii) Service provided:

1) Application to end user. The input would be selecting the treefruit, type of suggestion and location. The output is visualized data and suggestions.

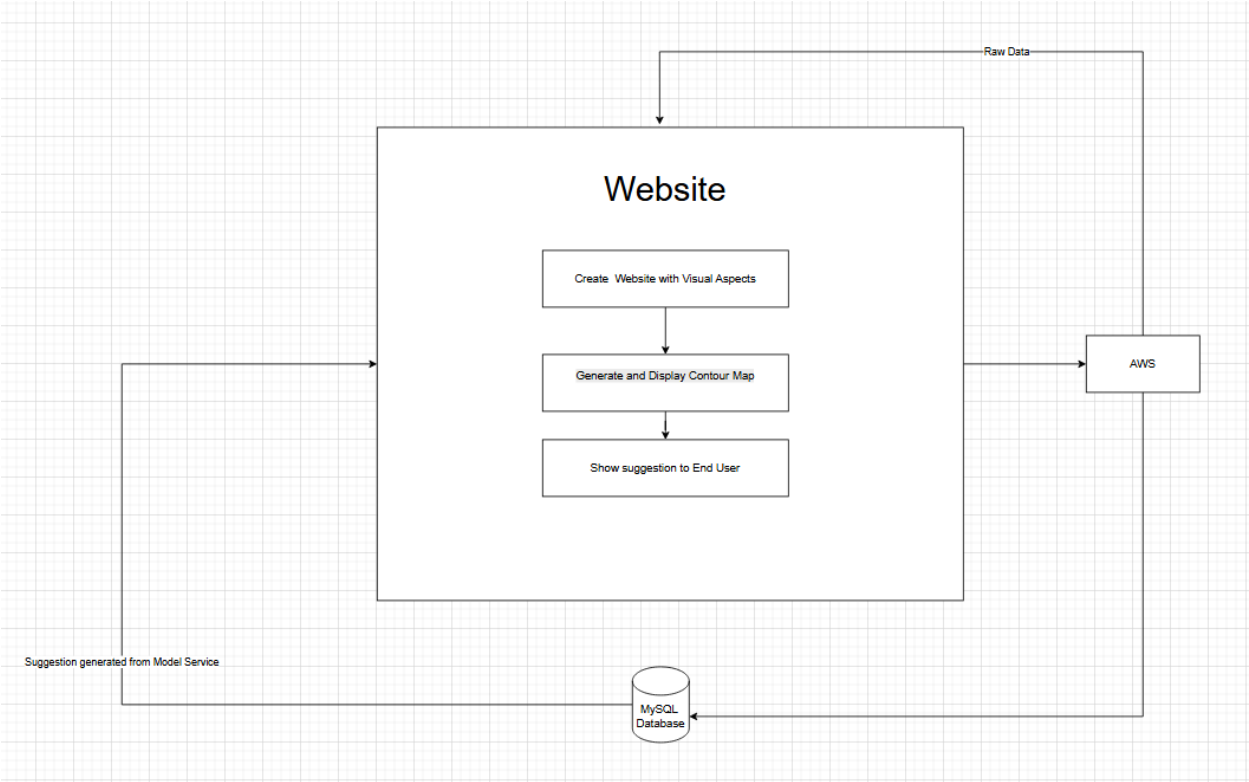


Figure 5: Website Service Visualization

VI. Data Design

The project has numerous data structures and uses a large and varied database to fulfill its functionality. The database contains entries for users, stations, and regions, which are stored when the user enters said information into the WSUDAS website. Users can customize and update these values to view data from their preferred stations. Insect, larva, and egg data is then used to determine when each pest will reach certain stages in their development, helping those in the agricultural sector decide whether and when they should take action to mitigate the potential damage those pests could cause. Users who opt to can also upload their data to aid the accuracy of the models, and all insect data is dated to allow users to look at the past to help them make better decisions in the present.

On the developer-facing side, the contours generated by the user's input are stored in the database with a unique ID and are associated with the stations used to generate them. These contours are stored in a PHP class structure with the associated stations being member values, and various functions are used in the process of retrieving and displaying the data to the user.

However, the vast majority of the data associated with the project is simply stored in a mySQL database and is retrieved and displayed to the user when prompted.

When generating contours, some data manipulation is involved. Each station acts as a point and has data for various insects associated with it. When generating a contour, the data for each station is gathered and then essentially “spread out” over an area depending on the density of each insect population at each station (The resulting contour often looks similar to Perlin noise generation with added variation to smooth out the visual disparities). The contour maps communicate the target region to the user through the shaded areas and then the densities of each insect’s population depending on the difference and intensity of the contour’s colors.

VII. User Interface Design

The user interface is viewed through WSUDAS’ site and gives the user the choice of what region they would like to see data from, what specific data they would like to view, and then displays a contour map given this input. The contours the user sees will only use the data from the stations they have specified. Users then choose what crops they would like to view (tree fruits, potatoes, etc) or which pests (each species’ larva, eggs, and adults).

Since our project is web based, the only installation required on the user-end is the station/traps setup if required. A typical user flow would be logging into the website to gain access to the website’s full functionality (this functionality is already complete and is thus not within the scope of our project), choosing their region, choosing the type of treefruit and pests for suggestion, and uploading the data in .csv format. The rest is taken care of in the backend. Then the suggestion tailored to them is shown on the website with a contour map to show the location the suggestion is applied to, which is the project’s core purpose.

VIII. Constraints

There are numerous constraints to take into consideration when working with the DAS system. As it is an existing live-service system, we do not have too much freedom to make changes to the overall structure since that would result in unnecessary downtime and relearning of the codebase which is something the client wishes to avoid. Furthermore, the Scoutlabs traps we are working with output data in a standardized format which is helpful, however, we are unable to change what data we are receiving exactly. Therefore we must work around these conditions when using the Scoutlabs data. When testing we need to keep these constraints in mind as there may be barriers we have to work around rather than through (like a disambiguation error we ran into when using the Scoutlabs trap data).

IX. Testing and Acceptance Plan

IX.1. Project Overview

The majority of our work over the course of this project has consisted of transferring data between existing locations within DAS' database, the majority of our testing involves ensuring that the correct values are moved to the right locations in the correct format. This necessitates having an understanding of how the tables in the database relate to one another and making sure that each step of the transfer is grabbing the correct information before committing any changes to the database. As such, we have been taking steps to verify that each step in this process produces the expected results.

IX.1.1. Test Objectives and Schedule

Our intent with the test plan is to ensure that all of the components we create for our client function as expected and will continue to function properly beyond the conclusion of the capstone class in Spring 2026. We expect to use a variety of testing implementations at different points during the project according to the needs of the current focus, including manual, integration, and unit testing.

As a significant portion of our focus has been dedicated to successfully transferring data from ScoutLabs traps to the models service in DAS, much of our testing involved checking that this data transfer process was functioning correctly. This primarily consisted of verifying the integrity of the data since several tables needed to be referenced properly in order to transfer the data into the destination table with the proper formatting. The simplest way to accomplish this was simply using php's dump() command and checking that the data was correctly formatted at each step during the transfer process. However, unit tests that simply confirmed that the correct data types were passed to the correct locations were also used to quickly verify smaller aspects of the code's functionality.

We expect to move on to UI components once the data transfer process is complete, which will be much harder to design unit tests around and as such will likely require manual testing to ensure that UI components are working as intended. However, our current understanding of this component is that information will be brought in from the database during this process, which we could likely design more formalized unit tests around.

The final deliverables of our unit tests will likely take the form of php's native testing framework for the vast majority of cases, which offers the ability to design unit test cases much like in a C# or Java environment. For any UI components we encounter we will utilize JavaScript unit tests to test the functionality of the user-facing web application, and those will be included in the final deliverable tests as well.

IX.1.2. Scope

The purpose of this section is to provide insight into our current methods and practices for testing our code to ensure that the work we produce will function as expected when deployed in a live environment.

IX.2. Testing Strategy

Our planned testing strategy will primarily incorporate a CI pipeline, as continuously deploying code without verification of its proper functionality is not acceptable for our client (they are running a live service and thus cannot afford to spend the time needed to roll back their site due to faulty code). With this method our code will be automatically tested when pushed while still

giving us the flexibility to postpone certain elements that are not yet working without requiring a full reboot of the system.

IX.3. Test Plans

IX.3.1. Unit Testing

As per the norms of unit testing, we will pass any standalone functions through a unit test to ensure that they are performing as we expect in an environment where they will not have an impact on the database should something unexpected occur. Since most of our methods currently involve grabbing data from one table and transferring it to another, there is not much to test during this process besides ensuring that the correct data appears where we expect it to.

IX.3.2. Integration Testing

Integration testing has been an important aspect of our project thus far, as there are numerous individual, compartmentalized pieces in DAS' system. The database is split across the docker containers, each containing dozens of tables used to store the data the system depends on. As such, it is important to guarantee that these separate pieces interact correctly with each other.

IX.3.3. System Testing

For what we have produced so far, black box testing would be directly comparing data processed with expected data. The tables in service-weather would be populated with sample data and the data inserted into service-models will be put into a test-case to compare with expected data. The whole DAS system is an extensive project with many parts we don't have to work on, so a full black box testing is deemed unnecessary. During the black box test, the first action is syncing the trap data into location, then, the location would be used as a reference point for syncing the records into the trap counts, the result would be two fully populated table location and trap counts

IX.3.3.1. Functional testing:

Functional testing would mean testing the functionality of the program. For our case, it would mean testing the first data transfer, the referencing for the second data transfer, and the second data transfer. At each step we verify that the data is of the correct type and value during the transfer.

IX.3.3.2. Performance testing:

We have already begun design implementations that keep stress-testing in mind, which includes using chunked inserts to the database so that the service does not run out of memory. We also check to see that our inserts run in a reasonable amount of time. The stress test is also minimized by the impact of chunking, so having one is not needed.

IX.3.3.3. Standards and Constraints Verification / Testing:

For this type of testing, the flow would be the same as UAT, so users only need to do one under the developer's supervision.

IX.3.3.4. User Acceptance Testing:

Since our work is mostly done on the back end of the project, UAT cases for this project would be having users go into the website and try out the features, which is not limited to what we work on. The users would go to the website, try to create the location and insert the trap data manually, and let the system do the work. The result is a contour map.

IX.4. Environment Requirements

The testing environment requires access to DAS' system through local docker containers that can run the system separately from the main site. Additionally, we use php's unit testing (and likely JavaScript's in the future), of which the libraries are built-in. NPM also needs to be installed and run through the web container to properly view the frontend of the site for any testing that must occur there.

X. Alpha Prototype Description

X.1. Contour Location/Traps Transfer

X.1.1. Functions and Interfaces

The transfer function uses the trap location data and creates new contour regions to be used by the contour system to generate new contour maps for the user-facing maps on the website.

X.1.2 Preliminary Test

In our testing we have discovered some hurdles with the system, specifically that multiple traps use the same location tag but are not in fact in the same location. This issue is caused by a trap user owning multiple traps, and thus we had to discover a method to disambiguate between them. We found that there was a unique attribute associated with each trap and decided to use this to differentiate the traps. While this issue has been resolved, we also must change our current implementation to use an update & insert method since this command will be regularly run and must not overwrite existing locations.

X.2. Scoutlabs Data Transfer

X.1.1. Functions and Interfaces

This is another transfer function which moves relevant data from weather to models in the DAS system. It is dependent on the transfer above as it must associate trap counts with a contour location to function properly on the live system.

X.1.2 Preliminary Test

While simply moving data to where it should be was not too much of a challenge, there are additional considerations we need to take into account when writing this function. Specifically, we need to refactor our current transfer to append a "survey_date" value to the entries so that they can be properly and easily used by the user-facing frontend service to display the data in chronological order.

X.3. Transfer Job

X.1.1. Functions and Interfaces

This function's purpose is to run a job to handle the two transfer functions listed above. It should call both of them in sequential order (contour, then traps) to properly insert data into the Models system at specified intervals.

X.1.2 Preliminary Test

We have not yet fully implemented this function as we want to ensure our above functions are working properly. An infrastructure for running jobs already exists within the system, so applying this aspect should not be too challenging going forward once the above functions are working smoothly.

XI. Alpha Prototype Demonstration

For this current cycle, the demo prototype would be a quick demonstration of data transfer from one service to another, namely Service-Weather to Service-Models. The trap records and locations are pulled from a Scoutlabs API, then stored into two tables: scout_labs_records for the actual data recorded from the smart traps and scout_labs_traps for the location of each trap. Then the data is migrated into Service-Models to contour-locations for the trap locations. For the contour_trap_counts, the data from scout_labs_records are iterated through two loops: a location loop for each location in contour_locations corresponding to the region_id and the insects in model_card, and an inner loop to push the newest record every week from the start date til now into contour_trap_counts.

XII. Future Work

Our goals for the next semester are to begin working on frontend implementation, specifically the "click on map" feature which would allow users to view detailed data at specified locations. This will be a significant departure from what we have been working on so far as it will involve more JavaScript writing as opposed to solely php, and we still need to clarify details with the client as to what exactly this will look like in its final form (i.e. would users be able to click anywhere in the contour region or would they be able to click on specified locations based on available stations.

XIII. Glossary

Entomology - A branch of biology focused on the study of insects.

API - (Application Programming Interface) Protocols that determine how applications interact with one another.

GDP - (Gross Domestic Product) The total value of all goods produced and services within a country.

WSUDAS - (Washington State University Decision Aid System) The organization running the DAS service, which provides agricultural information based on the organization's models.

Perlin Noise Generation - A type of pseudo-random visual generation that produces areas of various visual intensity.

XIV. References

[1] S. Zahniser, “What is Agriculture’s share of the overall U.S. economy?,” *Economic Research Service*, Dec. 19, 2024. [Online]. Available: https://www.ers.usda.gov/data-products/chart-gallery/chart-detail?chartId=58270

[2] *Decision Aid System*, “About | Decision Aid Systems,” *Decisionaid.systems*, 2025. [Online]. Available: https://decisionaid.systems/about. [Accessed: Sep. 13, 2025].

[3] The Codeholic, “Laravel 12 in 11 hours – Laravel for Beginners Full Course,” *YouTube*, Oct. 14, 2024. [Video]. Available: https://www.youtube.com/watch?v=0M84Nk7iWkA

[4] C. M. Bishop, P. Carlsson, S. Andersson, D. Mohamad, P. Nyman, and K. Naylor, *Pattern Recognition and Machine Learning*. MTM, 2018.

[5] R. Touti, “Perlin Noise: A Procedural Generation Algorithm,” *rtouti.github.io*, 2025. [Online]. Available: https://rtouti.github.io/graphics/perlin-noise-algorithm. [Accessed: Oct. 20, 2025].