Final Software Design Document

Version: 2.0

Date: February 14, 2019

Team Name: IntelliChirp

Project Sponsors: Colin Quinn and Patrick Burns

Team's Faculty Member: Fabio Santos

Team Members: Steven Enriquez, Michael Ewers, Joshua Kruse, Zhenyu Lei

Table of Contents

Introduction

It is becoming ever more important to track and manage the biodiversity that lives on Earth. More and more animals and plants are becoming extinct everyday which can create a major impact on other parts of the ecosystem. The group that our team is involved with, Soundscapes2Landscapes, is a science-based project looking to further advance biodiversity monitoring in order to save the lives of plant and animal species. Biodiversity is the study that ”refers to the variety of living organisms on Earth, how they relate to each other, their ecological function, and genetic diversity. All aspects of biodiversity are intimately linked to the functioning of ecosystems, where species interact with their physical environment. Biodiversity plays a vital role in many ecosystem functions, such as clean water, clean air, nutrient cycling, food production, and responses to disturbances, such as fires.” [1] It is vitally important to conduct proper biodiversity monitoring, in order to understand the ever changing environments that humans share with plant and animal species.

Our clients Colin Quinn and Patrick Burns are part of the Global Earth Observation Dynamics of Ecosystems Lab (GEODE). Colin Quinn is a PhD student and Patrick Burns is a Research Associate at Northern Arizona University. They work with Soundscapes2Landscapes to help achieve their biodiversity monitoring goals. Our clients use a specific type of monitoring called passive acoustic monitoring (PAM). This process allows more spatially extensive and continuous metrics for biodiversity. In Sonoma County, PAM has the ability to provide land managers and users with a better idea of animal species affected by development and conservation efforts. Our clients have assigned our team with the task of automatic sound identification from a soundscape, as currently sound identification is conducted in a manual time consuming way.

Problem

To understand the problems of the current implementation, the workflow process will be discussed. First, soundscape recording data is collected from low cost audio recording devices that are placed in different landscapes across Sonoma County, California. Once placed, these devices record one minute of every ten minutes for three to five days at each site. This has so far resulted in a total of over 500,000 minutes of gathered audio data. The soundscape recording data then moves onto sound analysis where biodiversity can be identified along with the specific layers of biophony, geophony, and anthrophony. Once the identification and analysis are done, satellite imagery from the International Space Station is used to create visual representations of the surveyed sites. Finally, the satellite data and sound data are put together to create a species distribution model, which can be used to track locations of bird species and potential environmental changes in their ecosystems.

The part of the process we are involved with is the soundscape manual analysis. Currently researchers would manually listen to audio files and draw boxes around various sounds. For a one minute clip, a researcher must listen to the clip, determine what the sounds are, draw boxes, and label each box with the corresponding audio component that is occuring. With noisy files, a researcher may spend over a minute going through a single file. This tool is useful but takes too long for scientists to effectively research the biodiversity in Sonoma County, California. Because of the frequent recording of audio, the researchers have resulted in terabytes of sound data for each individual site. Additionally, our clients would like for volunteers, or citizen scientists, to be able to analyze their own files. For example if a volunteer is working out in the field and records some audio, there is no current way for this volunteer to analyze their file as most of the features of the current identification tool are closed to volunteers.

Overall, the problems include:

- The manual identification process is very time consuming. Terabytes of audio is collected from each site, and requires people to listen to the audio and manually draw boxes around the sound components being searched for.
- Current interface is not easily accessible to volunteers. Soundscapes2Landscapes wants this tool to be able to be used by anyone, and the current interface is not very easy to navigate for non-tech volunteers.

Solution

Our solution to these problems will be an application called the Soundscape Noise Analysis Workbench. This solution involves a user-friendly user interface that hosts a machine learning model. The goal of this application will be to allow any user to upload their audio files for analysis.

Overall, our solution will consist of:

- User-friendly web application.
- An ability to automatically classify different audio components in the inputted file using the following machine learning models:
  - Support Vector Machine
  - Neural Network
- Calculated acoustic indices (data statistics used by sound researchers) for each uploaded sound file.
- Visualizations of the analyzed audio components.
- Table of audio components and acoustic indice values.
- A way to export each models' classification and the calculated acoustic indice values.
- A standalone version of all the features of the web application for offline use in the field.

The solution will ingest audio files. Researchers from Soundscapes2Landscapes will use audio files that they collect with the low-cost audio devices being used around Sonoma County, California. We will be using a machine learning algorithm to automatically classify different types of sounds in these recordings. Our machine learning model plans to accomplish the task of identification in a fraction of the time than current implementations, classifying an audio file in under a few seconds. The machine learning model requires training to accurately classify audio components. We must train the model on previously classified audio data. This previously classified audio data consists of many audio files that have been labelled with each category of sound that we are looking for. The sound categories will include birds, cars, rain, wind, and others. Collecting data to train the machine learning model will come from multiple sources, including open source data and data from Soundscapes2Landscape's audio recorders. The results of this classification will be visualized in a variety of ways. The visualizations will include a labeled spectrogram, showing the classified components in the inputted audio, as well as a pie chart of the proportions of each sound category. These categories include geophony, biophony, anthrophony, and a no sound present category. Additionally a table of all the information collected from the analyzed file will be displayed. The solution will also be created as an application for offline use in the field. This application will provide users the ability to classify their audio without an internet connection.

Requirements

From developing our planned solution to the problem our clients presented us with, we have created specific requirements that our envisioned solution needs in order to properly create a working final product. Our team has determined that this project contains key domain level requirements: users will be able to upload audio files, then analyze the uploaded files, then see the results of the analysis visualized in a timely manner, and then export all results.

From these key domain level user requirements, key functional requirements for the system were created. These requirements detail the specific features the product will provide our users when accessing the envisioned application. The application ingest audio files, then using machine learning will classify the sound components in the uploaded file, then calculate acoustic indices, then display the results in multiple ways, then export these results, and finally an offline version of the application for fieldwork is needed. In addition to the specific functional requirements our product will need to solve our clients goal, specific non-functional performance requirements were needed for our product to succeed. The performance requirements include: uploading a one minute file should take less than 5 minutes, and should only take at most three seconds to complete a full analysis. As well as specific performance requirements our team realized one important environmental constraint featured in our planned system. Our clients work currently only consists of data from Sonoma County, California. Since the data only consists of Sonoma County data, it is not easy to guarantee a high accuracy from our machine learning model when data from other parts of the world is to be used with our application.

After discussing with our clients the problem they are facing, working to find a envisioned solution to solve their problem, and creating the key specific requirements; a specific design must be created in order to finalize how the final product will be built. The purpose of this document is to outline the specific design of how the software will be built by the team. This document will explain the overall architecture of the final product, how each module of our solution works and how each module works with each other. Finally, we will explain how our team plans on implementing each part included in this document.

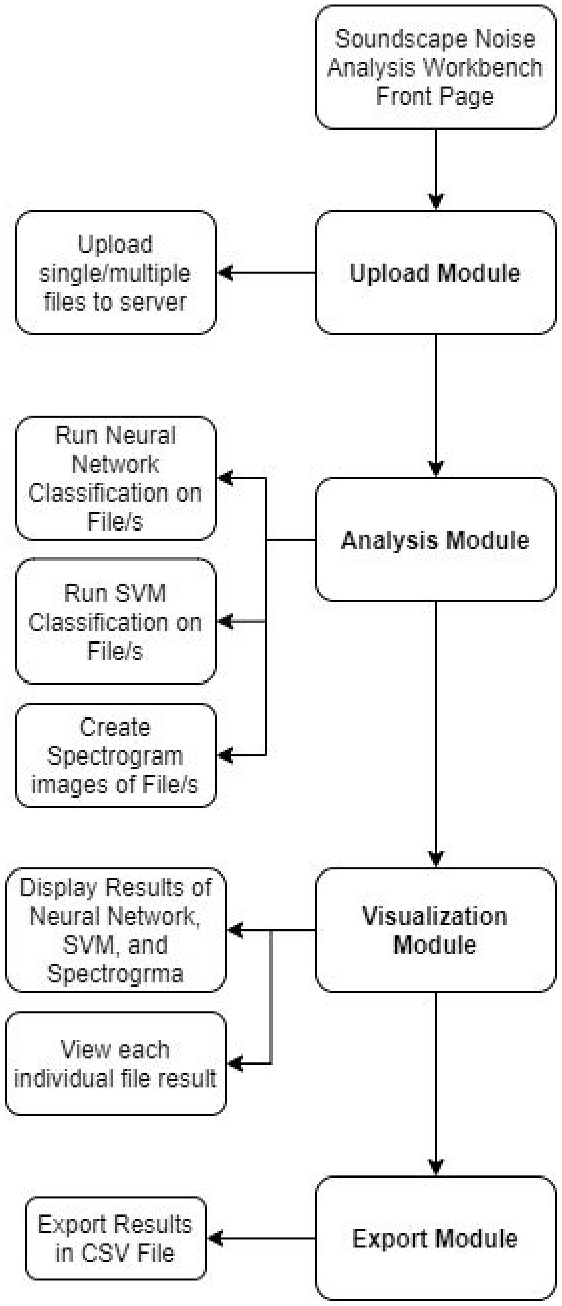
Implementation Overview

Through our requirements acquisition, our clients helped to reveal the major problems which we are looking to solve. The main problems which arose through our acquisition consist of the following:

1. The clients currently use manual identification when analyzing their large storage of audio files, which has proven to be very time consuming.
2. The current program, Arbimon, is not easily accessible to the client’s volunteers.
3. The current analysis does not run efficiently on an HPC.

The solution we have envisioned for our clients is an application which will be named “Soundscape Noise Analysis Workbench” (S.N.A.W). The S.N.A.W will consist of the following attributes:

Figure 1: Diagram of the Soundscape Noise Analysis Workbench System Components


Soundscape Noise Analysis Workbench Front Page
Upload Module
Upload single/multiple files to server
Analysis Module
Run Neural Network Classification on File/s
Run SVM Classification on File/s
Create Spectrogram images of File/s
Visualization Module
Display Results of Neural Network, SVM, and Spectrogrma
View each individual file result
Export Module
Export Results in CSV File

- It will be a user-friendly web application and an offline application.
- It will utilize a machine learning algorithm which can automatically classify specific audio components which are being searched for.
- The online web application will return clear visualizations of the analyzed audio file.

Our solution will require that audio files are input in the WAV format, as the client's audio recording devices collect and store data in WAV format. Once the files are uploaded, the application will then use a machine learning algorithm to start the identifying process. The usage of a machine learning algorithm to identify sounds within an audio file solves problem (1) stated above. We plan to create a simplistic design for the web application to ensure that it presents a user friendly experience while maintaining full functionality while analyzing the audio files, creating a simple design solves problem (2) stated above. Lastly, the creation of an offline application will directly solve problem (3) listed above, as it will be able to use the same functionality as the web application without the visualizations. The offline application will be more in tune for analyzing bulk files on an HPC.

The technologies we have chosen for the solution consist of the following: React, JavaScript, Flask API, and many Python libraries. Each of the technologies listed will contribute and work together to create our product. React will be used to create our web application, and allow us to create a user-friendly front-end. The Flask API will be used as the "glue" to connect our React front-end to our machine learning scripts. The Flask API will also be able to run the React front-end as a server which will receive API calls and present different pages through URL requests sent from the front-end. These technologies will be able to produce the tools we will need to finish the product according to how our clients expect it to be. To view more details on information for the specific Python and React libraries we have chosen to work with, please refer to the Technologies Appendix.

Architectural Overview

In the previous section, we discussed the implementation overview that our team has developed to produce our product. To understand the architecture of our system, we will discuss the high-level detail on how the Soundscape Noise Analysis Workbench will be built. Below are multiple diagrams of our system components [Figures 2,3,4].

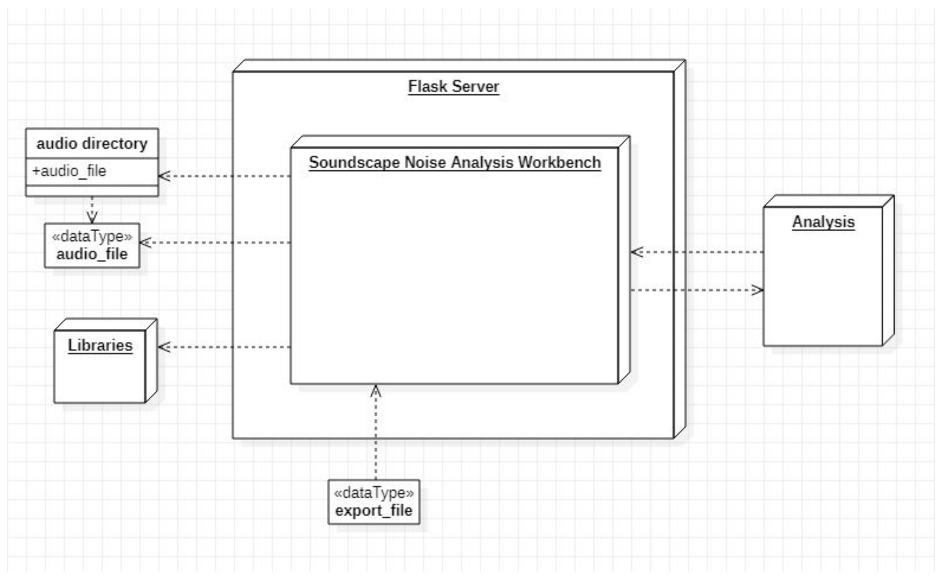


Figure 2: Diagram Overviewing the S.N.A.W Server Architecture

An outline of our Soundscape Noise Analysis Workbench server architecture is shown in Figure 2. A Flask server runs the S.N.A.W. module. The server takes in inputs of either a directory of audio files, or a single audio file. The server connects to the libraries needed by each component of S.N.A.W. The server sends data and receives data from our analysis module. Finally the server sends an exported file to the user.

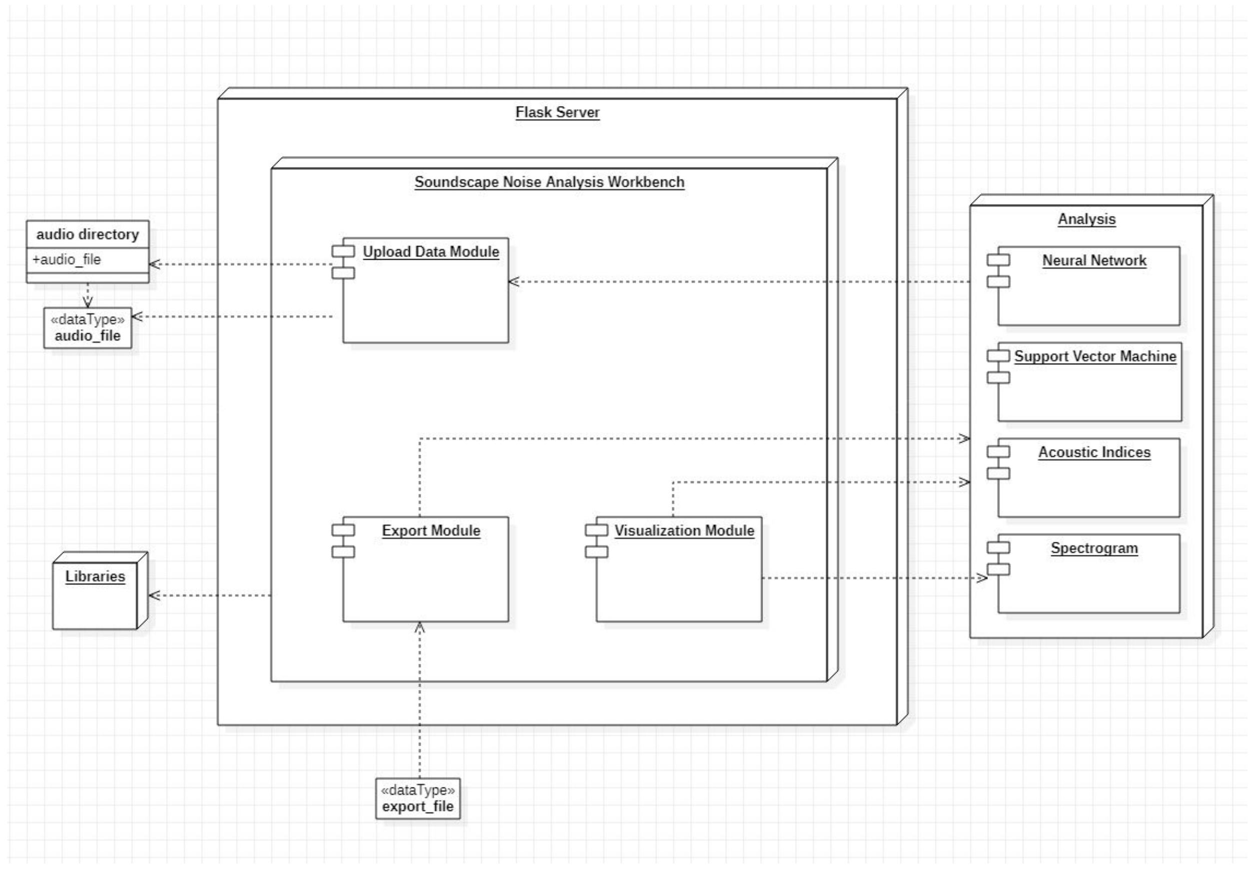


Figure 3: Diagram of the S.N.A.W. Architecture's Modules

An outline of the Soundscape Noise Analysis architecture modules is shown in Figure 3. Taking in the inputted directory of audio files or a single audio file, the upload data module will handle the data and send the server locations of each file to the analysis module. The Analysis module uses the uploaded files and runs the Neural Network, Support Vector Machine, Acoustic Indices, and Spectrogram Modules. These modules return the

Data Structure as described in “Data Structure Description” to the Export Module and Visualization Module. The Export Module sends an export file to the user.

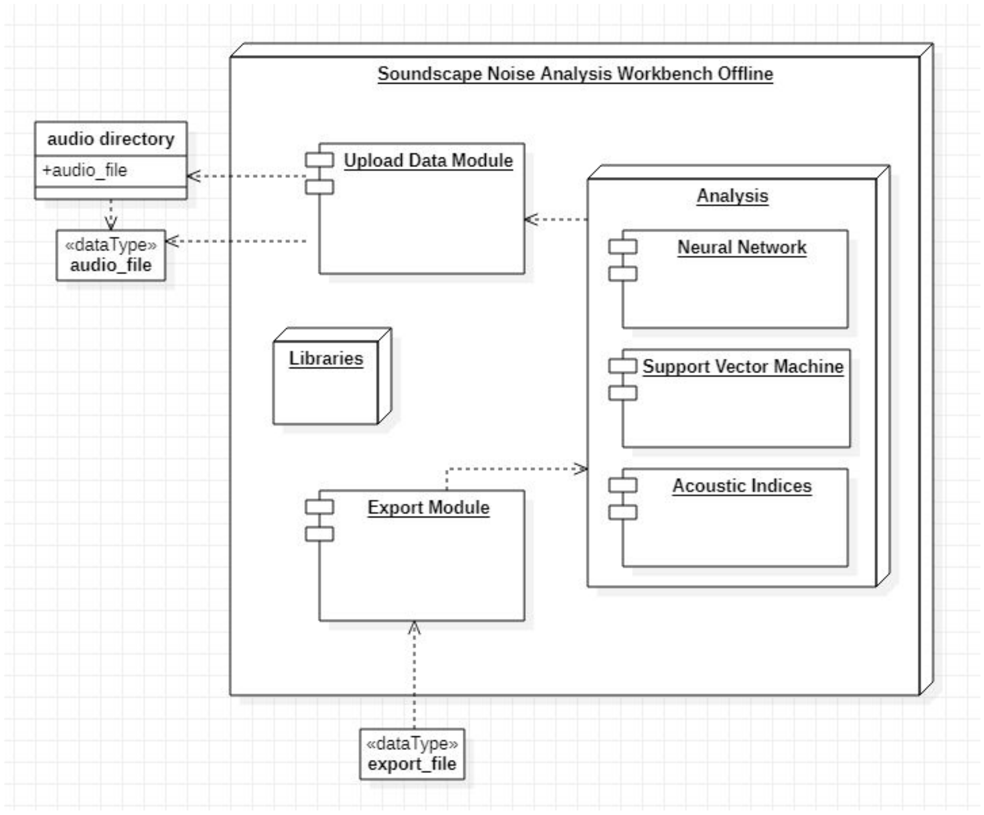


Figure 4: Diagram of the Standalone Offline S.N.A.W. Architecture

A diagram of the Standalone Offline S.N.A.W architecture is shown in Figure 4. The component modules of the system work very similarly to the web application with a few differences. Firstly the workbench does not sit in a server but is instead an standalone executable file. The libraries needed by each component are included in the file itself instead of being called by the server. Each module found in the analysis module is included in the offline file itself instead of being called by the server. Additionally the Spectrogram Module and Visualization Module are not included in the standalone application as the only output to the user is the exported file.

Our product, the Soundscape Noise Analysis Workbench, will be developed in Python and Javascript. Python will be utilized for the back-end of our system with the microframework Flask. Our front-end will be developed using React, a Javascript library. Our offline version of our application will run as a single python script. Below we will overview the key responsibilities and features of each component of our system, which include:

Upload Data Module

The Upload Data Module will be utilized to ingest audio files that will be analysed. The application ingests files in WAV format and stores them for further analysis. The user will be able to choose to analyze a single file or multiple files. This will be done by dragging and dropping the file(s) onto the web application, or by choosing the files from a file chooser.

Analysis Module

The Analysis Module is used to take the ingested audio files and run multiple types of classifications on them. This includes running the audio files through a Neural Network to identify sound components, using a SVM to identify sound components, and run Acoustic Indices calculations.

Export Module

The Export Module will allow a user to export the analyzed results. The user will then be able to keep a log of the results on their local machine. The results will be in the CSV files. There will be a CSV file for the Neural Network classification, a CSV for the SVM classification, and a CSV for the Acoustic Indices calculations.

Visualization Module

The Visualization Module is used to visualize the results in a user-friendly manner. The user will be able to get an intuitive visual of what sound components were present in the file, where they were found, as well as how big of a proportion of the audio file was identifiable sound components.

Standalone Offline Script Module

The Standalone Offline Script Module is used to have a standalone version of the Soundscape Noise Analysis Workbench. This will be in the form of a script that can be run through a terminal. Using a standalone version of the application will be useful for anyone looking to analyze audio without a connection to the internet. The user will need to provide a path to the a directory of audio files to be analyzed.

These modules make up all of the functionality that SNAW will provide. Now we will look into the communication mechanisms and information flows of our architecture. The web application connects the Upload Data Module, the Analysis Module, the Export Module, and the Visualization Module. The Upload Module will ingest the audio files and store them in a location that the Analysis Module will pull from. The Export Module will input the data from the Analysis Module and allow for the results to be downloaded. The Visualization Module also uses the Analysis Module's output to create user-friendly visualizations for the end-user. The Standalone Offline Script Module is separated from all of the other modules. The control flow will be in a script that runs each classification offline.

With the communication mechanisms and information flows discussed, we will discuss the influences from our architectural style embodied by our architecture. For our React App, we have modularized the application into many components. This allows for updating the code base in the future and debugging the application much easier.

Module and Interface Descriptions

In the previous section, we discussed the high-level details on how the Soundscape Noise Analysis Workbench will be built. To understand the lower-level details of our system's architecture, we will go into detail for each individual module.

Upload Data Module

This module is used to ingest WAV files into the system. Files will be selected from a file chooser or can be dragged and dropped onto the web application. An error will be shown if the user attempts to upload a file that is not accepted by the web application. The ingested files will be uploaded to a server to then be utilized by the Analysis Module. The Upload Data Module sits at the very beginning of our products architecture, as files are needed to move forward with the execution of the product. Below is a UML diagram of the Upload Data Module [Figure 5].

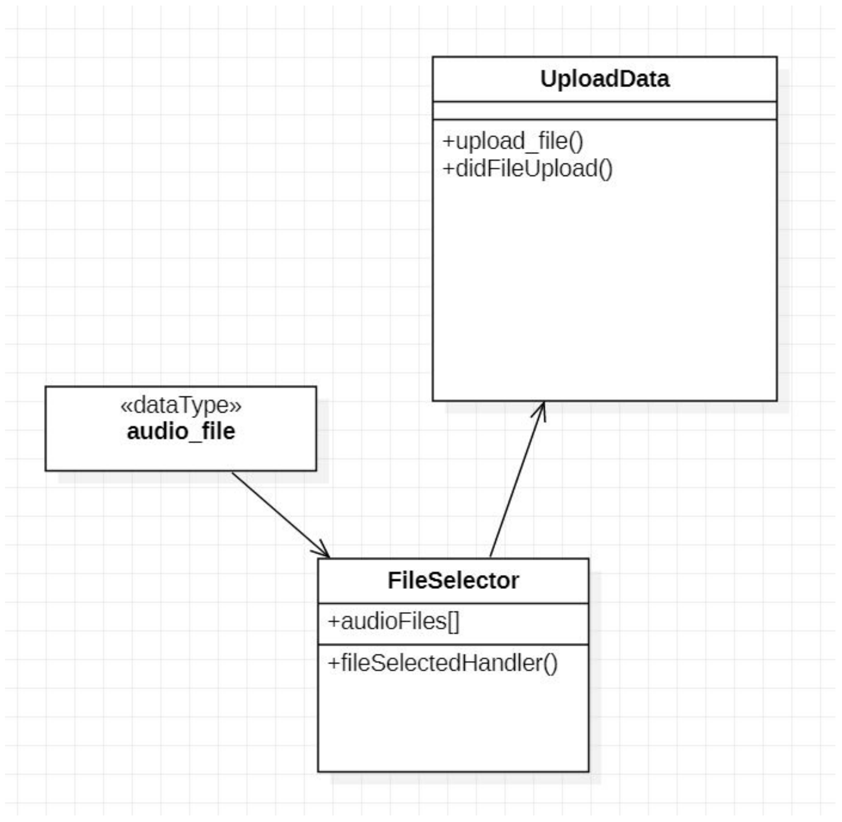


Figure 5: UML diagram of the Upload Data Module

Input = A WAV file or multiple WAV files (unless we choose to accept other popular file formats).
Output = An array of files uploaded to the server with a confirmation message to the end-user.

Analysis Module

This module will be used to run Neural Network classifications and Acoustic index classifications on the uploaded audio file(s). Once the Upload Module has completed, the user will be able to press the "Analyze Audio" button which will run 3 seperate classes and their methods to analyze the audio. The SVMClassification and NeuralNetworkClassification classes will run our machine learning models on the uploaded audio file/s and return a JSON dictionary populated with the results from each of the classes. The AcousticIndices class will run many different methods to calculate the specific acoustic indices within the file, and then return a JSON dictionary populated with the result data. Upon completion of all three classes running successfully, the Analysis Module will then send a populated JSON dictionary to the Visualization Module to be processed and displayed on the front-end of the product. The results from the Analysis Module will also be sent forward to the Export Module. The Analysis Module is an important part for the overall product, as this is where the main portion of calculations on files is done for the product. Once completed, the product is then accessible through the Visualization and Export Modules. Below is a UML diagram of the Analysis Module [Figure 6].

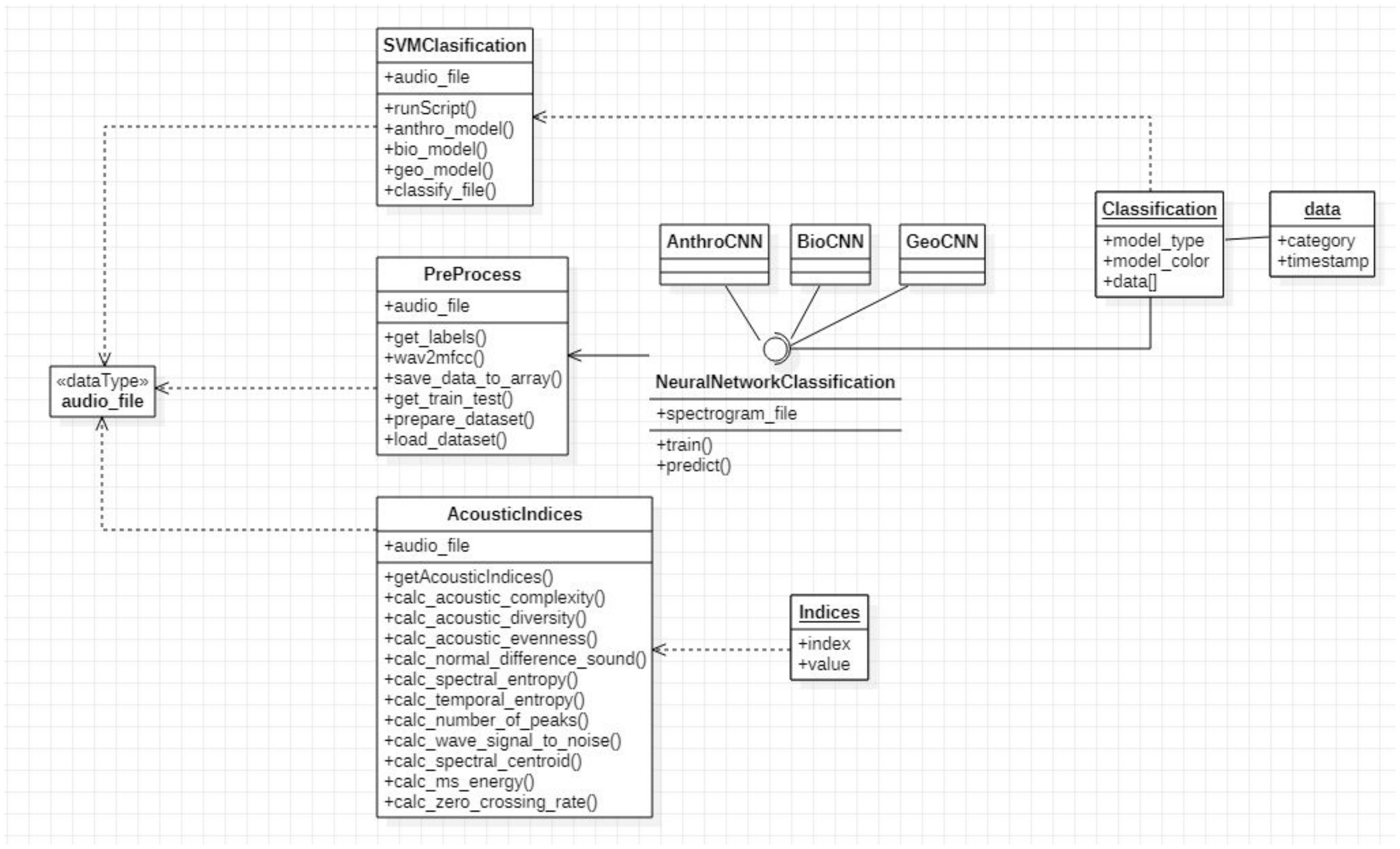


Figure 6: UML diagram of the Analysis Module

Input = Audio File(s).
Output = JSON Dictionaries from each classification containing the results.

The Analysis Module's output references a data structure known as a JSON Dictionary, which will be one of our main data structures organizing our resulting data. JSON dictionaries utilize Key-Value pairs, in which a unique "Key" (string, integer, etc) is stored and will be attached to specific data or "Value" that we choose. We pass a single JSON object to the front end. This overall object contains a JSON object for each uploaded file. Associated with each uploaded file is a JSON object with the results of each of the four different back-end analysis functions. The UML Diagram below showcases the design that we have decided on for our JSON dictionary [Figure 7].

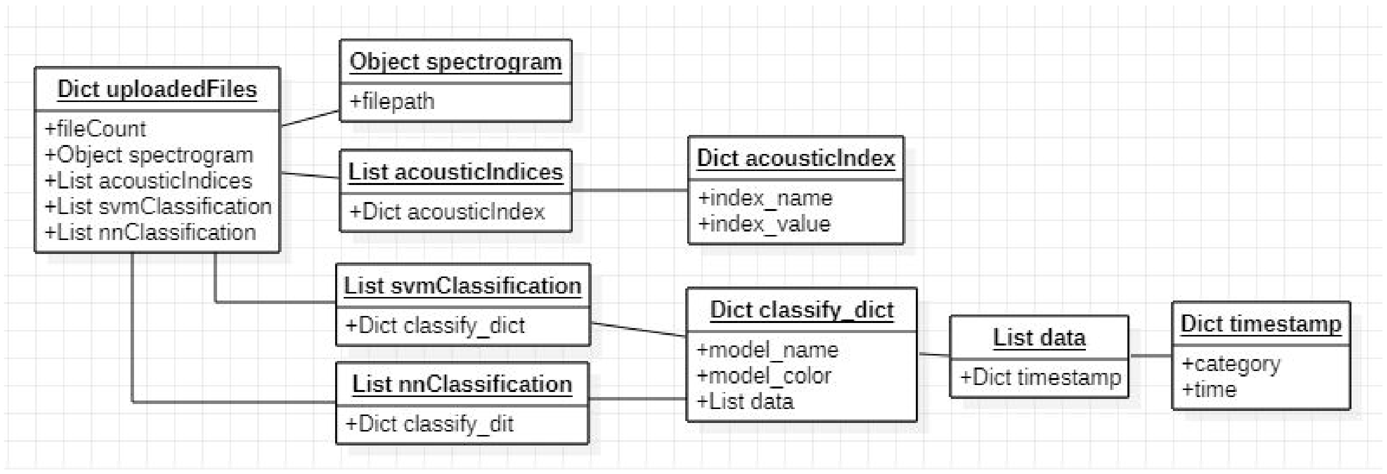


Figure 7: Diagram of the JSON Structure

Visualization Module

This module is used to visualize the data needed by users in a user-friendly way. It accepts the results generated by classification Python programs and acoustic indices Python programs from the back end. The Visualization Module will generate a certain number of extension panels imported from material UI, depending on the number of files uploaded by users. We have ReceiveAndCalculate sub-module, which receives the dictionary that is passed by the Analysis Module running neural network and acoustic indices calculation. The dictionary contains three categories: anthrophony, biophony and geophony. The visualization module also have another GenerateGraph sub-module, which will calculate the number of these three categories respectively, and then generate the line chart and pie chart through the line chart and pie chart files imported from recharts. They all include the charts imported from the rechart package. This sub-module also takes the spectrogram from the results folder and displays it in cardmedia. A line chart is displayed as a scalable vector graph, and a pie chart is also a scalable vector graph. All the data in the table are from our calculation of classification dictionary and acoustic index dictionary. Here is the UML diagram of the visualization module [Figure 8].

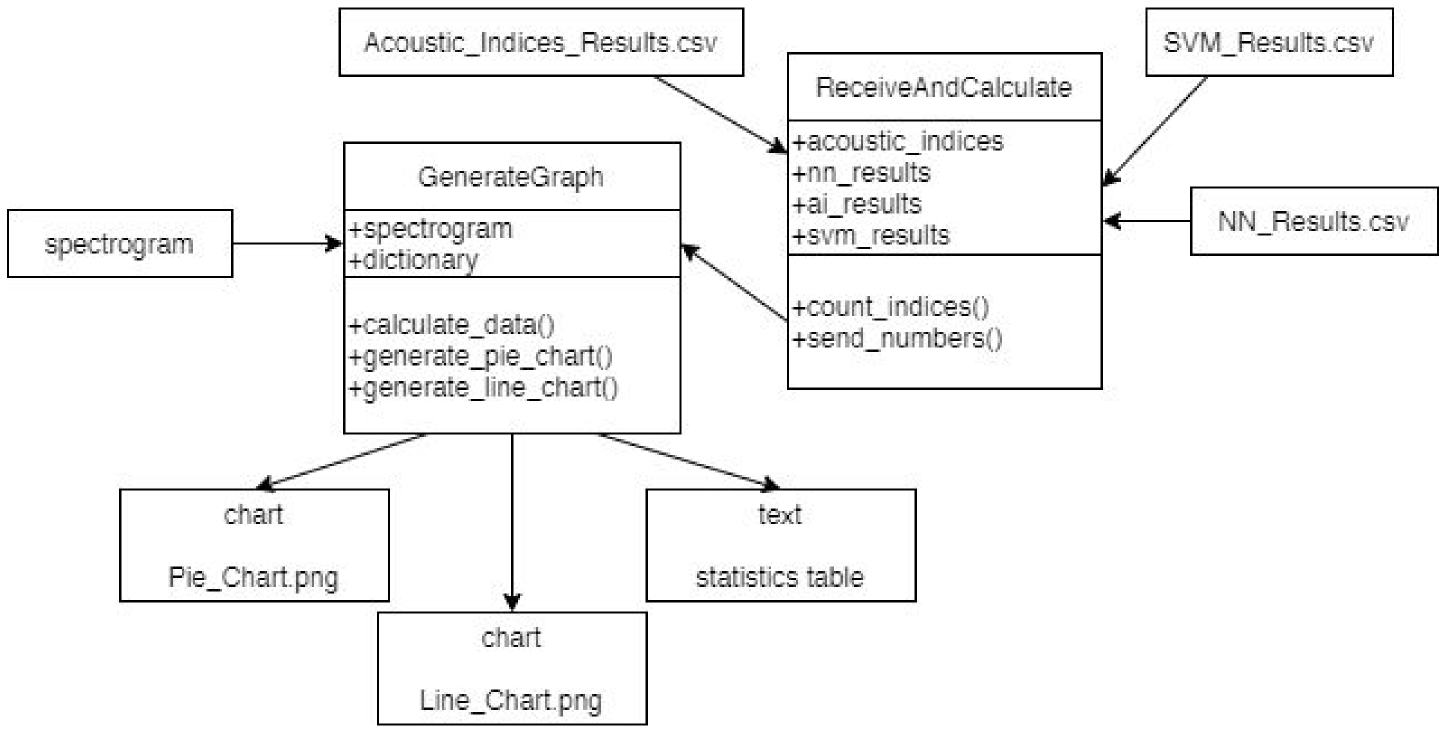


Figure 8: UML diagram of the Visualization Module

Input = Dictionary.
Output = Spectrogram, Charts, Table of values of each of the acoustic index percentage.

Export Module

This module is used to export a CSV file that has been populated with result data from the backend. The Export Module will receive a JSON dictionary which is passed through by the Analysis Module which runs the Neural Network and Acoustic Index calculations. The Export Module will be run by the ExportResults class, which contains methods for the Neural Network Classification, SVM Classification, and Acoustic Indices calculations. Using the data received from the JSON dictionary, the selected CSV of the three options mentioned will be formatted properly. The NN_Results.csv will contain data regarding the specific sound events that were identified within the audio file, along with the respective timestamps at which the sound events occured. The SVM_Results.csv will contain similar data to the NN_Results.csv, except that the data will be obtained from running an SVM model instead of a Neural Network model. Lastly, the Acoustic_Indices_results.csv will contain data on the acoustic indices of the specific file analyzed, and will include each of the sub-categories in the acoustic indices and their respective values calculated. The Export Module is found at the end of our products architecture as its main objective is to allow the user to retrieve the results of the product after the Analysis Module has been completed. Below is a UML diagram of the Export Module [Figure 9].

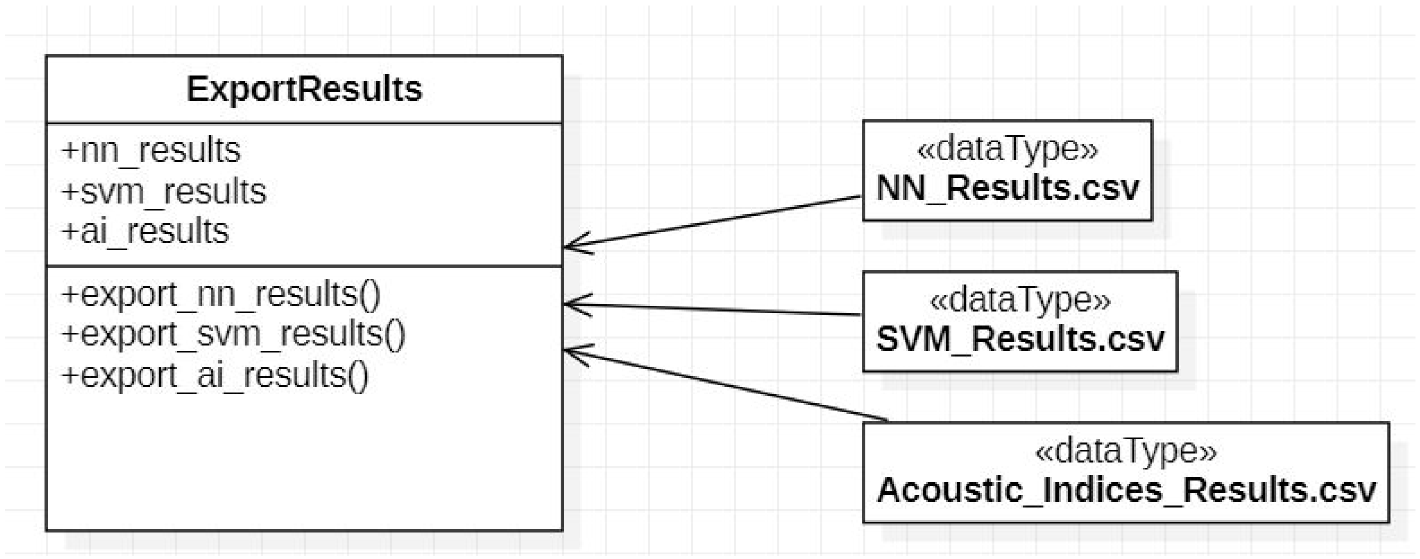


Figure 9: UML diagram of the Export Module

Input = JSON Dictionaries from the Neural Network Classification, SVM Classification, and the Acoustic Indices calculations.
Output = Data populated CSV files for each of the classifications and calculations.

Standalone Offline Script Module

This module will be used to run the the classifications and Acoustic Indices calculations on the inputted files through a command line interface (CLI). The classifications will include the Neural Network as well as a SVM model. The models are pre trained by the team and are ready to classify audio components. This offline script will allow users to use this model without a connection to the internet. This offline script will also allow users to classify audio files through a high performance computing (HPC) cluster. Our clients are looking to classify audio files on Northern Arizona University’s HPC cluster Monsoon. Below is a UML diagram of the Standalone Offline Script Module [Figure 10].

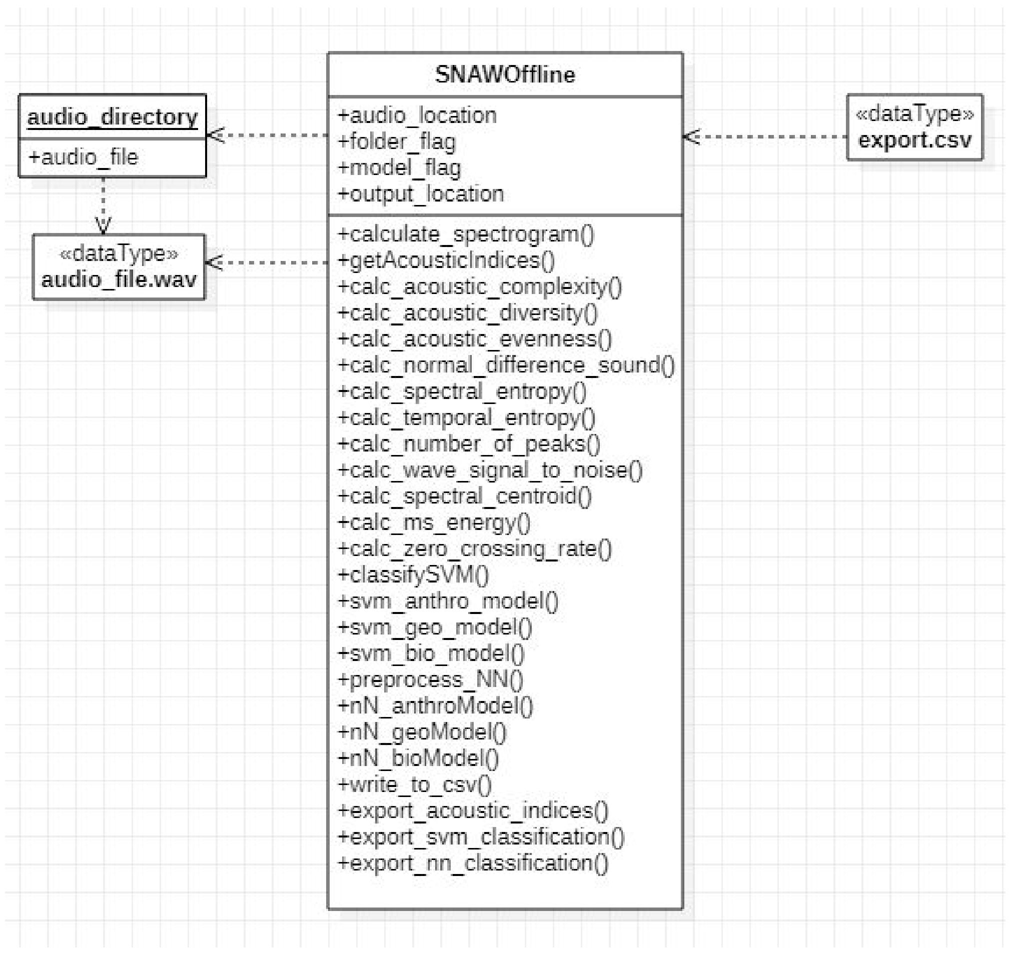


Figure 10: UML diagram of the Standalone Offline Script Module

Input = File path to a single or multiple WAV files to be analysed.
Output = The results of the Neural Network classification, SVM classification and Acoustic Indices calculations as individual CSV files.

Implementation Plan

Now that we have discussed in detail the modules and interface descriptions, the focus will shift into the implementation plan. In this section, we will have a timeline of different implementations that will lead us to the final product. Now that we have the minimum viable product, which can upload, submit and output an analysis of audio files given by a user, we will be iteratively improving the Neural Network as the semester continues.

Below are the major phases we will have in this semester in order to be successful in creating our product.

Phase 0: Finish the Minimum Viable Product of each Component

Currently our application is able to upload WAV files with the Upload Data Module. The files are analyzed with a basic Neural Network, an SVM model and Acoustic Indices calculations. The data can be exported as a text file, and there is no standalone version of the application currently. Our first phase is to fully implement a working minimum viable product. This involves having a Neural Network trained on data that we have collected, being able to export results as CSV files, and having a basic running standalone version of the application.

Phase 1: Application Optimization and Further Development

Our local application has been tested and proved to be working. With this as our base minimum viable product, we will create more branches for additional features and improvements, so that we can continuously improve our products to meet the final requirements of clients. This will include improvements to the web application and the standalone offline script.

Phase 2: Improving and Retraining Neural Network

We will be improving our Neural Network by adding more data to the training dataset as well as tweaking the Neural Networks parameters. In parallel with the above phase, during this phase, each member of the group will be working individually. This work will include labeling and classifying work with the data our clients gave us. With the data labeled and exported, this will give us a bigger and better training data set. We can use that to train our Neural Network more until Spring break.

Phase 3: Front-End Improvements

Currently, our user interface has a certain number of expansion panels, depending on the number of files uploaded by the user. In each of them, we will first display the spectrogram of the audio file(s), and then we will show the big picture of the combination of a line chart and pie chart showing the result of sound classification and the prediction of sound components. Finally, we will have a table showing sound types and proportions. We will talk to our clients and iteratively improve the user interface and each component..

Phase 4: Implementation

With these in place, our team will begin to complete our final local application. We will put together the application of all the completed modules, and only need to merge together to ensure that it can run on the computer's local server. After that, our team will implement our finished local product to the school's high-performance computing cluster, which will not consume much time.

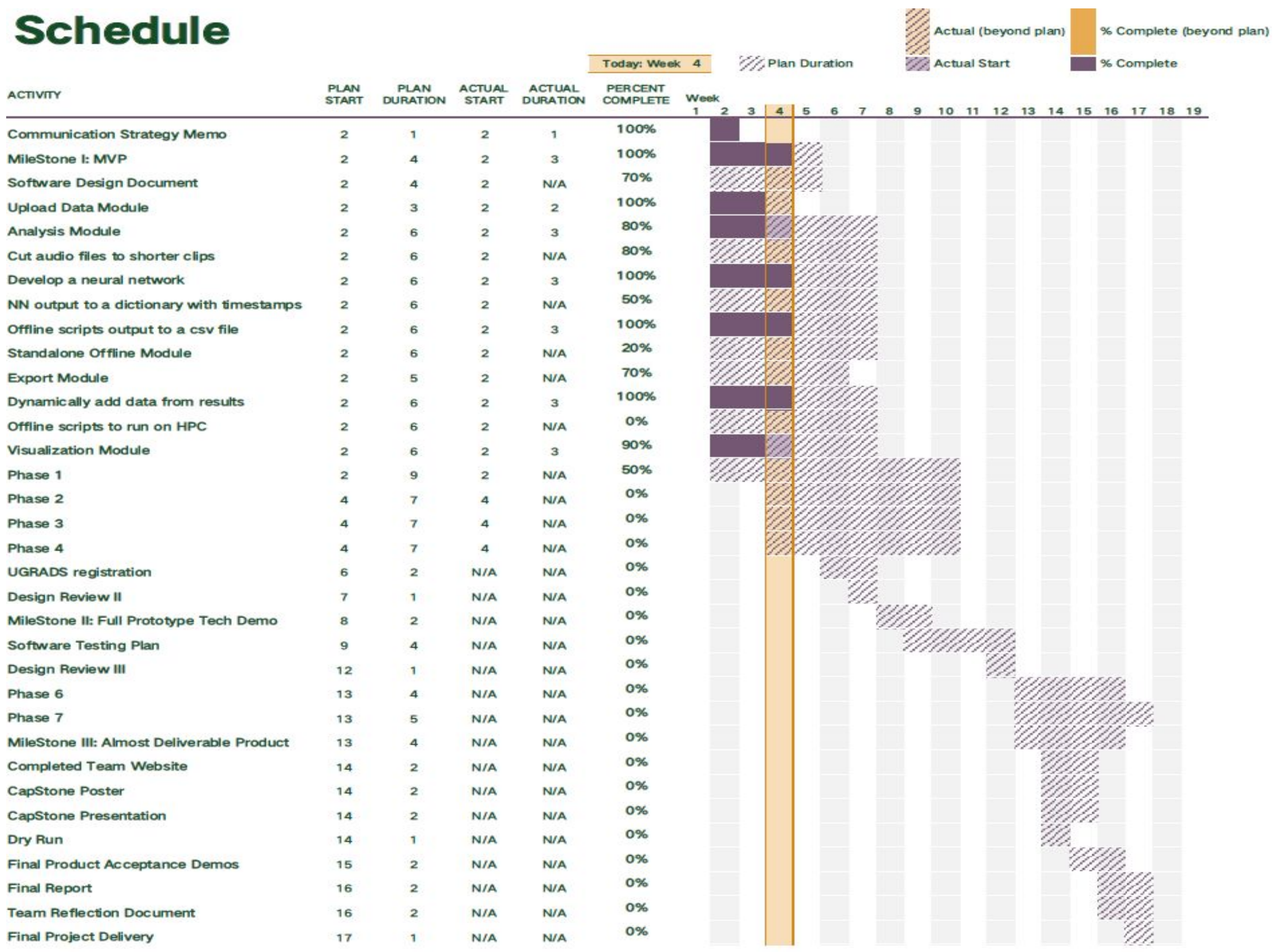
Phase 5: Testing and Bug Fixing

Once we have implemented Phase 4 and secured the product on NAU’s Monsoon HPC, we will rigorously test and debug the program. We will upload audio files in different formats, corrupted audio files, and run other tests that may cause the program to crash. Then we will implement the necessary functionality to solve those bugs. This way we can ensure that our application runs smoothly and reliably.

Phase 6: Official Release

At this point, our project is done, and we will formally deliver the product to our clients. We will meet with the clients and introduce the final version of the product step-by-step. We must ensure that the client is able to effectively utilize our product. We will be sure to include a user manual with detailed information on how the product can be executed properly to our clients.

Below is a Gantt chart showing when our team plans on completing the implementation of each module, the testing, as well as when we will complete the integration between components [Figure 11].



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Communication Strategy Memo	2	1	2	1	100%
MileStone I: MVP	2	4	2	3	100%
Software Design Document	2	4	2	N/A	70%
Upload Data Module	2	3	2	2	100%
Analysis Module	2	6	2	3	80%
Cut audio files to shorter clips	2	6	2	N/A	80%
Develop a neural network	2	6	2	3	100%
NN output to a dictionary with timestamps	2	6	2	N/A	50%
Offline scripts output to a csv file	2	6	2	3	100%
Standalone Offline Module	2	6	2	N/A	20%
Export Module	2	5	2	N/A	70%
Dynamically add data from results	2	6	2	3	100%
Offline scripts to run on HPC	2	6	2	N/A	0%
Visualization Module	2	6	2	3	90%
Phase 1	2	9	2	N/A	50%
Phase 2	4	7	4	N/A	0%
Phase 3	4	7	4	N/A	0%
Phase 4	4	7	4	N/A	0%
UGRADS registration	6	2	N/A	N/A	0%
Design Review II	7	1	N/A	N/A	0%
MileStone II: Full Prototype Tech Demo	8	2	N/A	N/A	0%
Software Testing Plan	9	4	N/A	N/A	0%
Design Review III	12	1	N/A	N/A	0%
Phase 6	13	4	N/A	N/A	0%
Phase 7	13	5	N/A	N/A	0%
MileStone III: Almost Deliverable Product	13	4	N/A	N/A	0%
Completed Team Website	14	2	N/A	N/A	0%
CapStone Poster	14	2	N/A	N/A	0%
CapStone Presentation	14	2	N/A	N/A	0%
Dry Run	14	1	N/A	N/A	0%
Final Product Acceptance Demos	15	2	N/A	N/A	0%
Final Report	16	2	N/A	N/A	0%
Team Reflection Document	16	2	N/A	N/A	0%
Final Project Delivery	17	1	N/A	N/A	0%

Figure 11: Gantt Chart displaying the team's schedule

As we can see in the Gantt Chart, we are currently on the right track, and our planned tasks are gradually being completed. We have completed a major functional module, that is the Upload Data Module, because it is less complicated than other modules. For the Analysis Module, we are running a Neural Network and an SVM model to analyze audio. As we are labeling and categorizing more data, we will update our Neural Network and SVM model to improve accuracy. The Standalone Module, Export Module and Visualization Module are the three modules we are still improving. Among them, the Standalone Module has relatively less to do related to the functionality of the web application. There is no need to upload files because the standalone script will only need a file path to the audio files. We are doing well in the Export Module, and we can already export a SVM classification text file, but we want to output a CSV file with a unified standard output format. The Visualization Module is one of the closest modules to be completed. We are displaying a spectrogram, a line chart, a pie chart and a table of values of each of the acoustic indices for each uploaded file in a expansion panel. Currently the Visualization Module is in a tentative state, as we may change the way that the product will display the resulting data.

Right now, the Upload Module is done, and four of us team members are in charge of the remaining four modules. Joshua is developing front-end interface, therefore, he is responsible for the visualization module, and also some options such as changing the layout according to the needs of the customer. Zhenyu is doing the Standalone Module, which provides another version to customers, so that people can no longer analyze audio files by uploading files and submitting, but can directly select local file paths to analyze files. Steven and Michael are developing our neural network instead of CityNet or SVM we used, so the Export Module and the Analysis Module are in their hands.

Besides all of the documents, we will keep our track on Phase 1, 2, and 3, and we plan to complete Phase 4 before Spring Break, which is to complete the cluster implementation to ensure that our program can run on Monsoon after successfully running on a local server. After a few design reviews, some dry runs with mentor, finish the poster and presentation, we will enter the final stage of the project, which is a complete report and delivery to the user. We are confident in this and we strive to make a useful and reliable application.

Conclusion

The impact of human involvement on ecosystems has major consequences. More than one million plant and animal sciences are going extinct, with the majority happening within the last few decades (IPBES). There is an ever growing need to properly monitor the biodiversity in ecosystems, as well as the factors that impact biodiversity. Our clients Colin Quinn and Patrick Burns work with the science-based group Soundscapes2Landscapes to provide a more effective and efficient way to monitor biodiversity. They have tasked our team with building an automatic way to identify specific, individual sounds present in a soundscape recording.

The problem that we are trying to solve is the time-consuming manual identification process of audio components in recordings from various sites in Sonoma County, California. Our solution is to develop a user-friendly web application that hosts a machine learning model to automatically classify these audio components. This will allow volunteers as well as researchers to efficiently classify the components of their recorded soundscape files. As a stretch goal, we will also be implementing an offline fieldwork application for use on a laptop in order for researchers in the field to also use our application. In order to get one step closer in making this solution a reality, we have put together this technological feasibility document.

This document's goal is to outline the specifics of how our team will develop our envisioned solution. Using the requirements developed in our Requirements Document, our team has developed a software design that accomplishes all the requirements our solution must meet in a user friendly and timely manner. This document outlines each specific way all the modules included must be implemented and the ways each module will interact with each other. By following this document our team is confident that we can provide our clients with a user friendly solution that will solve all aspects of the problem outlined in our introduction.

Appendix A

Needed Libraries for Front-End and Back-End

Python Libraries (back-end)	Reason
numpy	Calculates different acoustic indices
librosa	Helps with loading uploaded audio files
peakutils	Detects peaks in uploaded audio files
SimpleITK	Spectrogram image processing
scipy	Identifies components found in an audio file
os	Helps with loading uploaded audio files
flask	Allows back-end python files to connect with front-end
werkzeug	Helps with loading uploaded audio files
matplotlib	Plots spectrogram image from uploaded audio file
base64	Helps with saving spectrogram image file
pyAudioAnalysis	Support Vector Machine library used for classification
wave	Helps with loading uploaded audio files
contextlib	Helps with loading uploaded audio files
keras	Toolkit for Neural Network classification
wandb	Visualizing Neural Network classification
sklearn	Splits train data into train and test datasets
tqdm	Progress bar for Neural Network console logging

React Libraries (front-end)	Reason
material-ui	React framework for user interface
jquery	Allows front-end to connect with the back-end server
serviceWorker	Offers performance advantages to the React application
Recharts	Creates chart components for visualization of data