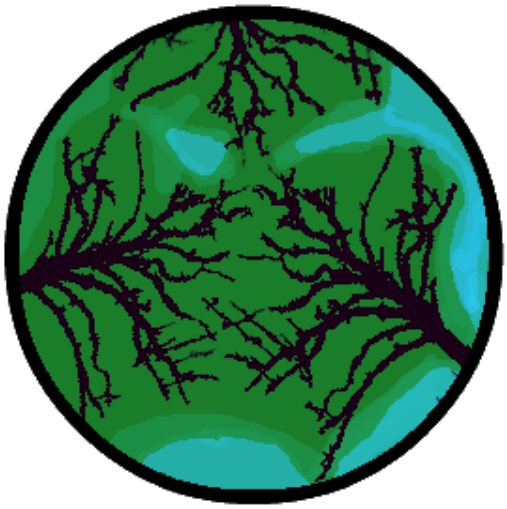
Software Design

February 7th, 2020

Canopy - Team 11

Team Members:
Robert Plueger
Dongyu Xia
Maria Granroth
Sponsor: Dr. Patrick Jantz, Ph.D.
Mentor: Scooter Nowak



Version 1.4

Table of Contents

1 Introduction

We are Canopy, a 2019 - 2020 Northern Arizona University Computer Science capstone team. Our team members are Robert Plueger (team lead), Maria Granroth, and Dongyu Xia. Our project is to develop an application for characterizing 3D vegetation structure in tropical ecosystems. We will analyze the data obtained from the Global Ecosystem Dynamics Investigation (GEDI). GEDI data is collected via lidar, laser ranging of forests and topography. The data consists of the 3D distribution of stems, branches, and leaves, making up the vegetation structure of ecosystems. GEDI is an Earth Ventures mission that started in December 2018 and is funded by NASA. GEDI will acquire billions of vegetation profiles across the Earth's temperate and tropical ecosystems in two years.

The sponsor and client of this project is Dr. Patrick Jantz. He is a member of the Vegetation Structure as an Essential Biodiversity Variable (VSEBV) project based at NAU. He is also a team member funded by the NASA Applied Sciences program. He will use GEDI data to develop vegetation structure essential biodiversity variables (EBVs). EBVs can be used by policy makers and scientists to improve land use decisions and guide priorities for conservation of biodiversity in tropical landscapes.

The reason for this project is that current workflows are clumsy and require a lot of manual work. Our client needs to use the R language to process a large amount of GEDI data repeatedly, and the workflows require that users know how to code. The primary goal is to create an application to help improve processing speed and accuracy while reducing repetition and manual steps. We envision an application that supports both researchers and policy makers while making the analysis process easier on people who do not know how to code. High quality analysis from our web application will make it easier for our end users to make decisions for conservation and resource management.

This application allows users to log in and choose a region to analyze. When the user selects an area, the software will acquire GEDI data and analyze it. The application automatically generates the desired analysis for users to view and

download. Our application is expected to process 50,000 files in an hour. If the user selects an entire country or continent, the processing time is expected to be up to 3 days. Our application plan is written in Python. The computation and website will run on the web hosting service Microsoft Azure, so our application's primary environmental constraints will be the computation speed of the server and the price of the service.

2 Implementation Overview

This section will discuss our project solution and the tools we will be using to aid us in our project. We will discuss how our solution works and how each tool will help us in our goal.

Our project solution is to create a website that can analyze GEDI data and return data of the user's choosing. We have four main components that will assist in the creation of our product.

We must be able to publicly host our product. We are using Microsoft Azure to host our website. Not only will Azure be able to make the website available for public use, it will be able to store data and allow for backend calculations. We will be allotted monthly speed and storage.

We are required to compute a large amount of data, and we are using Python to compute the necessary operations. To assist with the combination of Python and HTML, we are using Flask, a web framework that utilizes Python. This allows us to produce analytic results, read files, and deliver files through the backend of our product.

We will also be working with uploaded shapefiles. Our primary operation on shapefiles will be checking the validity of their formats. This is necessary because we are utilizing shapefiles to extract data from NASA's Application for Extracting and Exploring Analysis Ready Samples (AppEEARS) database. We will be using GeoPandas, a Python package involved with shapefiles, to help check that any uploaded shapefile is formatted correctly.

Additionally, we will be visualizing shapefiles. Along with GeoPandas, we will be using Folium, a package interface of Leaflet, a mapping software. Folium is

compatible with Flask, and will allow us to easily map shapefiles and allow users to manually create shapefiles via a map.

3 Architectural Overview

This section will discuss what our system will do in order to produce the desired analysis for the user. We will indicate what modules will be included, list what their responsibilities are, and describe how information will flow from one module to the next.

Diagram 1 demonstrates our system's architecture. Every module pictured will be a component in the system. Blue boxes indicate modules that are frontend and can be interacted with by the user via the user interface. Yellow boxes indicate modules that are backend and will not be visible to the user while the application is performing those steps.

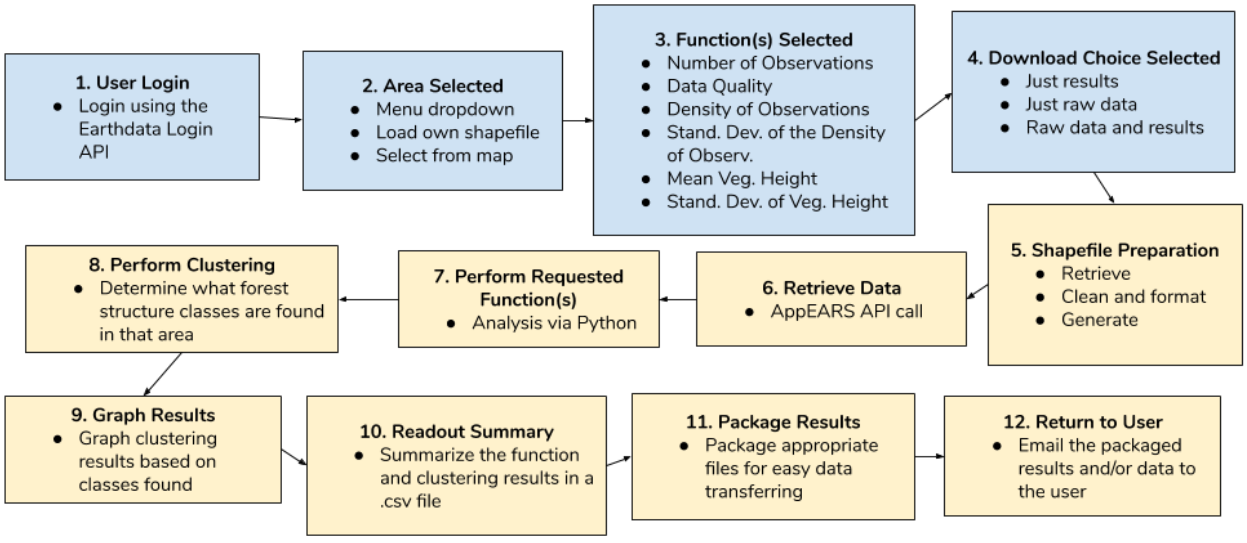


Diagram 1 – Architectural Diagram

3.1 Discussion

The style used to build our system's architecture was loosely based off of the standard model-view-controller (MVC) pattern commonly found in web applications. As demonstrated in *Diagram 1*, our system can be broken into two primary sections- the frontend and the backend. The frontend acts as the "view" portion of the MVC pattern, where the user can interact with the application, while the backend acts as both the "model" and "controller" portions of the pattern, where data is processed.

Moving forward, we will list the responsibilities and features of each module and describe how the individual modules will communicate and connect. Flask will be the most helpful technical component in assisting us with communication between the backend and frontend. Using Flask's framework, a button click on the user interface can be directly mapped to the corresponding functions in the backend.

3.1.1 User Login

- Login using the Earthdata Login API

The Earthdata Login API will be utilized to allow users to log in. Users need to do this in order to use our website. We will need this login information in order to retrieve data in **module 6** in *Diagram 1*.

3.1.2 Area Selected

- Menu dropdown
- Load own shapefile
- Select from map

The user will have these three options for selecting their area. Each area-selection option will be linked to different shapefile preparation methods in **module 5** in *Diagram 1*.

3.1.3 Function(s) Selected

1. **Number of Observations** - The total number of lidar pings that were performed in that area
2. **Data Quality** - The average quality of the observations

3. **Density of Observations** - The amount of observations per km2
4. **Standard deviation of the density of observations** - Standard deviation performed on the density of observations
5. **Mean Vegetation Height** - The average vegetation height found over the entire area
6. **Standard Deviation of Vegetation Height** - The standard deviation of the vegetation heights found over the entire area

As a baseline, the user will have these six options for functions. They may select one or many. More functions may be added later on.

3.1.4 Download Choice Selected

- Just results
- Just raw data
- Raw data and results

The user has these options for what they'd like to receive after our system is done analyzing their area. This information will be referenced in module 11 when the system packages the appropriate files.

3.1.5 Shapefile Preparation

- Retrieve
- Clean and format
- Generate

The shapefile preparation is dependent on what option the user used to choose their area in **module 2** in *Diagram 1*. If the user used the menu dropdown, then a preloaded shapefile will be retrieved from our system. If the user uploaded their own shapefile, then our system will clean and format the shapefile to be appropriate for AppEARS expectations. If the user selected an area via the map, then the system will generate a shapefile based on the coordinates selected. **Module 6** in *Diagram 1* will need the shapefile from this step in order to make the AppEARS API call for the data.

3.1.6 Retrieve Data

- AppEARS API call

In order to retrieve the data, the system will make an AppEARS API call using the user's login information from module 1 and the prepared shapefile from **module 5** in *Diagram 1*.

3.1.7 Perform Requested Function(s)

- Analysis via Python

Python math will be used to perform the functions requested by the user in **module 3** in *Diagram 1*.

3.1.8 Perform Clustering

- Determine what forest structure classes are found in that area

Three different clustering methods (hierarchical, k-means, and Python's hdbscan package) will be applied to the data to determine what different forest structure classes are found in that area. This will help visualize the quality of the habitat.

3.1.9 Graph Results

- Graph clustering results based on classes found

Using the information found in **module 8** in *Diagram 1*, the system will produce a graph of the clustering results based on the percentage of different classes found in the area.

3.1.10 Readout Summary

- Summarize the function and clustering results in a csv file

The system will print the various results from the analysis in **module 7** in *Diagram 1* and clustering in **module 8** in *Diagram 1* into a csv file for easy interpretation.

3.1.11 Package Results

- Package appropriate files for easy data transferring

The system will package files based on what the user chose for download results in **module 4** in *Diagram 1*.

3.1.12 Return to User

- Email the packaged results and/or data to the user

The system will email the package made in **module 11** in *Diagram 1* to the email address the user used to log in in **module 1** in *Diagram 1*.

4 Module and Interface Descriptions

This section will detail the functionality and connectivity of our product. It will present what each module does, and will lay out how each module interacts with others.

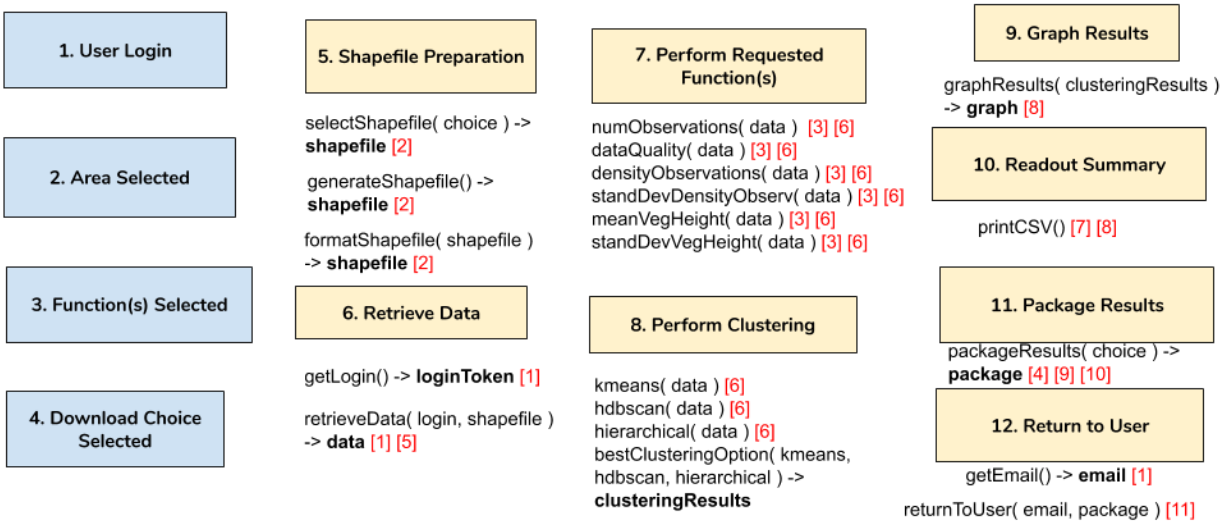


Diagram 2 - Function Sequence

Diagram 2 demonstrates the functions the system will be using in order to produce the end result. The functions that return something important to other functions indicate what they return by an arrow and a bolded variable name. The red numbers in brackets next to each function references the module number that the function is dependent on.

4.1 User Login

An Earthdata user login is necessary in order to retrieve GEDI data for **module 6** in *Diagram 2*. Earthdata provides a login API that we will use for this purpose. The API will help us retrieve a login token to use on the behalf of the user for the AppEARS API call to retrieve the data. The API will also be used to retrieve the user's email address to send them their data in **module 12** in *Diagram 2*. Since this module is frontend and HTML based, it does not have traditional functions.

4.2 Area Selected

The user will need to select an area to analyze. This can be done via our dropdown menu, selection on the map, or by uploading their own shapefile. This module is also part of the user interface, so there are no functions to describe. Each option to select an area will be mapped to an appropriate function in **module 5** in *Diagram 2*.

4.3 Function(s) Selected

The user can select one or many functions to analyze their data with on the user interface. Each button option selected here will be mapped to its corresponding function in **module 7** in *Diagram 2*.

4.4 Download Choice Selected

On the user interface, the user can choose to download just the results, the raw data, or both the results and the raw data. The option chosen here will be referenced in **module 11** in *Diagram 2*.

4.5 Shapefile Preparation

Depending on how the user made their area selection in **module 2** in *Diagram 2*, there are a few different ways shapefiles can be prepared for them in this module. If the user selected an area via the dropdown menu, selectShapefile will be called, using their area choice as a parameter. This function will grab a preloaded shapefile. If the user selected an area via selection on a map, generateShapefile will be called. This function will use Folium to write a shapefile based on the coordinates selected on the map. If the user uploaded their own shapefile, formatShapefile will be called, using the uploaded shapefile as a parameter. This function will clean and format the shapefile to fit WGS 84, which is the standard for geographic datasets. This function will utilize the Python package GeoPandas in order to do this.

The shapefile prepared in this module will be referenced in **module 6** in *Diagram 2*.

4.6 Retrieve Data

We will retrieve data from NASA's AppEEARS database via the AppEEARS API. We will get the user's login token from **module 1** in *Diagram 2* using the function getLogin. This will obtain the user's login token utilized by NASA's Earthdata login API. Then, we will request data from the database, sending the shapefile and the user's login token from **modules 1 and 5** in *Diagram 2* as parameters. This request will be sent using the function retrieveData. **Modules 7 and 8** in *Diagram 2* rely on this data to perform their operations.

4.7 Perform Requested Function(s)

We will perform the statistical analyses requested by the user in **module 3** in *Diagram 2*. The data from **module 6** in *Diagram 2* will be read and the following functions may be performed:

- numObservations - this function will analyze how many lidar observations were collected in the retrieved data
- dataQuality - this function will analyze the numeric quality level of the retrieved data

- densityObservations - this function will compute the number of lidar observations per unit of area
- standDevDensityObserv - this function will compute the standard deviation of the lidar observations per unit of area
- meanVegHeight - this function will compute the mean height of the observed vegetation
- standDevVegHeight - this function will compute the standard deviation of the height of the observed area

Each of these functions will return their results to a dictionary so that printCSV in **module 10** in *Diagram 2* may reference them when it generates the summary for the user. The dictionary layout may look something like this:

```
{ numObservations : <return into here>, dataQuality:
<return into here>, ... }
```

4.8 Perform Clustering

In this module, three different clustering methods will be applied to the data to determine what different forest structure classes are found in that area. The functions kmeans, hdbscan, and hierarchical will all be run using the retrieved data from **module 6** in *Diagram 2* as their primary parameter. In order to determine which clustering method was the most effective for that particular dataset, bestClusteringOption will compare the results from all three functions. Since we do not have a known cluster structure for these datasets, we will be using internal indices to measure the performance of each algorithm. The results of the best performing clustering algorithm will be what is referenced in **modules 9 and 10** in *Diagram 2.*

4.9 Graph Results

The function graphResults will produce a graph based on the clustering procedure done in **module 8** in *Diagram 2*. The graph will represent and illustrate the classes of forests in which different forest regions belong. This graph will be packaged in **module 11** in *Diagram 2.*

4.10 Readout Summary

Modules 7 and 8 in *Diagram 2* will produce a csv file that contains a summary of the analyses and clustering performed. The function printCSV will reference the dictionary that stores the results of all of the statistical functions that were run in **module 7** in *Diagram 2*, along with the results of the most efficient clustering algorithm found in **module 8** in *Diagram 2*. The results of printCSV will be packaged in **module 11** in *Diagram 2*.

4.11 Package Results

The analyses and clustering results, summarized in **modules 9 and 10** in *Diagram 2*, will be packaged to be sent to the user, shown in **module 12** in *Diagram 2*. The items packaged will be determined by the user's choice in **module 4** in *Diagram 2*. The function packageResults will perform this packaging.

4.12 Return to User

The user will receive their requested analytics back in the form of a csv, png, or ZIP file via email. First, we must retrieve the user's email from their account from **module 1** in *Diagram 2*. The user will then be sent the packaged items from **module 11** in *Diagram 2*. The function returnToUser will send the email to the user with the package attached. Note that the user will not see any data that is not returned to them via email.

5 Implementation Plan

Diagram 3 provides a detailed Gantt chart of our team's expected schedule. Before Week 4, each member needs to learn and be familiar with the tools we need. Then in Week 4, we need to complete the statistical functions we currently need, and link these python functions to our flask website. After that, we need to complete the shapefile preparation, and make sure our application can output the result of the data we get and package the results in Week 5. In Week 6, We will plan to complete user login, retrieve data, download choice selected and perform clustering. In Week 7 , we will complete the area selected and return the results to the user.

So far, most of the functions of our software have been realized. In the following time we will continue to communicate with our client and further improve our software. This schedule can help us to arrange and plan the time very well. It clearly shows when and what tasks we should complete. At the same time, it also sets aside a spare space to handle unexpected deviations in our plan.

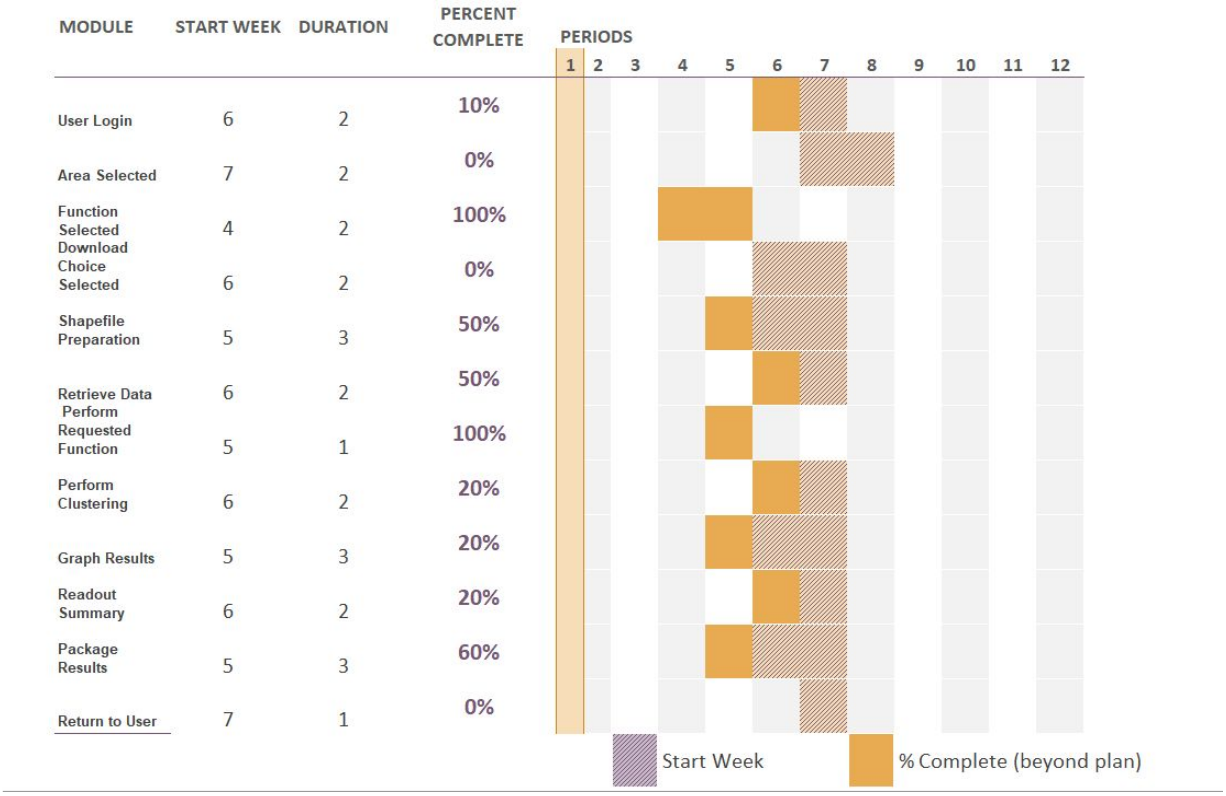


Diagram 3 - Gantt Chart for Spring 2020

6 Conclusion

This project is to develop an application to describe the three-dimensional vegetation structure of tropical ecosystems. Our application will analyze the data obtained from GEDI. The originator and client of this project is Dr. Patrick Jantz. He will use GEDI data to develop basic biodiversity variables of vegetation structures. Policymakers and scientists can use these variables to improve land use decisions and guide priorities for the protection of biodiversity in tropical landscapes.

Our application can help users analyze GEDI data. The user can **log in** to their Earthdata account and **retrieve data** from their account. Our users can **select areas** through a menu, map, or by uploading their own shapefile. After that, our application will **analyze** the data for the area selected by the user, and the results will be **packaged** and **sent** to the user.

So far, we have completed the compilation of statistical equations and successfully linked our equations to our website. We have also completed the acquisition of data and shapefiles. According to our schedule, our current development is in good condition. We are confident that we can complete the components laid out in our schedule. We also believe that our application will perfectly realize the requirements expected by our sponsor.