Final Report

04/25/2022

Team Biosphere

Project Sponsors: Jenna Keany, Christopher Doughty

Team Mentor: Melissa D. Rose

Team Members: Matthew Nemmer, Brandon Warman, Teng Ao, D'yanni Bigham

Table of Contents

Introduction

Tropical forests are vital to the global system and environmental problems cannot be overstated. The tropical forest absorbs CO2 and provides oxygen to promote the Earth's climate-positive recycling. Our clients Dr.Chris and Jenna Keany are working for NAU's Megabiota Lab where they are studying megafaunas such as animals that live in Africa, remote sensing, and Animal nutrient movement.

For our clients and their research group, they are facing difficulty in processing and visualizing the GEDI data which significantly helps them to observe mid-Africa climate change and use the information for ecologists to conclude. Moreover, the information and conclusions that they got from the GEDI data can help Policymakers to set up different policies to do with forest protection and preservation in the mid-Africa area.

As for Biomapper's potential users, our clients required us to create a mobile application that could significantly help our clients and their research group to view the mid-Africa map easily and without any programming skills. As required, the Biomapper will be used by our clients, researchers who are major in environment protection and preservation. Moreover, the application can be used by students and policymakers as well. In an easier to descript the user, it could be someone who wants to easily view tropical forest geographic data on a mobile device and use it whether for study purposes or observing purposes.

What we have created for our client's mobile application is called Biomapper, an Android application. There have some main features to support users to ease viewing different data offline /online. Moreover, it can filter the map by input values and set the interesting area to use often. It could significantly reduce the coding and processing time for datasets and give the user a simple and easy to view tropical data on a mobile device.

By achieving all requirements that our client is asked. There are several important parts for the team to establish.

Data processing and Map tiles creation

GEDI data is public on the website and we need to use Google Earth Engine to pre-process it and make the mid-Africa area separated by tiles in each zoom level.

Server construction

Amazon Server needs to construct map tiles and API features, it can support users access the map tiles and filter them from the mobile application.

Mobile Application Construction

Android application has the Google map methods to provide the underlying map and the application functionalities will be able to view the geographic map smoothly. Moreover that the application will have an About Page to introduce how to use it.

With the Biomapper completely and successfully finished, our client and their target user will have an easy-to-use and solid toolset to view the tropical forest data for the Mid-Africa area whether online/or offline. Using Biomapper to view the tropical forest map, will reduce the processing and programming time for the researcher who would like to view the data. As a result, the researcher and ecologists can spend more time collecting observations in the field.

Process Overview

The Biomapper team has four team members that take on different types of roles.

Matthew Nemmer: Team Lead
    Tech Lead, Front-End coder, Back-End coder.
Teng Ao: Release Manager
    Front-End coder.
D'yanni Bigham: Architect
    Back-End coder.
Brandon Warman: Recorder
    Back-End coder.

To create and deliver a complete solid mobile application the team is using the following tools to organize:

Github:
As the most popular open-source tool, the team is creating a Github Organization for Biomapper and will upload the Application source code with the developing process. The source code on Github will be updated when the team takes refinement or progresses.

Trello Board:
As a way to show which task is taken by who, the team will use the Trello board to distribute the tasks and show the current status of progression. Trello board will support a clear way to see team members' current tasks and also can avoid multi-take things happening.

Discord:
As a way to communicate with the team, the team is using the Discord communication method. There have several different channels for the team to find the meeting information and documents easily.

Zoom Meeting:
As the remote way to meet with team and client. The team is using Zoom to hold meetings every single week. Zoom support is a solid way to meet with others while they are out of town or can not make the in-person meeting. It makes the team and clients' communication better.

By using tools to organize tasks and assignments with all team members, we have a weekly routine team meeting with Mentor and client:

Routine Mentor Meeting.

For every week's Tue. 11 A.M the team will have a routine meeting with Mentor, a routine meeting was included completed tasks check out, in-progress tasks inspecting, and future tasks plan instruction. The routine meeting every week can make the team motivated and be clear about what to do for the next step. Also, the routine mentor meeting will help the team understand more about the Capstone schedule and will not overdue any upcoming tasks.

Routine Client Meeting:
Every week Friday at 11 A.M the team will have a routine meeting with the clients, a routine meeting with the client included current progress showing, plan to design, and project requirement changing. A routine client meeting will help the client know more about current progress and the team will also talk about the design requirements plan and see if those need to be changed. If the client has more functionality requests, the team will reconsider the design plan and try their best to make the new functionality come true.

Routine Team meeting:
For every week after 11 a.m.'s client meeting, the team will keep doing the team meeting, the team will talk about the next week's plan and decide who is doing what in that meeting. That will significantly help the team to distribute tasks and solve the remaining problem or confusion.

As previously talked about the way to organize the team and communicate with mentors and clients, below is the process overview that how we construct and create the Biomapper Application.

Data Acquisition:

The team was importing the raw satellite LIDAR dataset to Google Earth Engine, then export the datasets from Google Earth Engine and create map tiles for all datasets.

Front end:

The team is using Android Studio as a developing platform and Java language as the basic language to create the application. And with Google Maps Platform, the map will be
shown as a flat entire map by retrieving different zoom levels and <x,y> coordinates map tiles from the server.

Back end:

The team is using an Amazon EC2 instance as a backend server which is supported by clients. The team set Node.JS as an environment and helps the application request the

map tiles. Moreover, the Node.JS will run the filter python scripts while the application has a filtering request.

With the hard work of all team members and with all tools, meetings, and designing plans fully implemented, the team can create and complete an amazing project and deliver it to clients.

Requirements

The tropical forest is vital to the global system and environmental problems will never be overstated. The client is focusing on using high-precision data from satellites and processing it into data products for ecologists and researchers to locate specific animals in mid-Africa. And with the conclusion made by ecologists, their goal is to use the information to help policymakers to set up different policies do with forest protection and preservation in mid-Africa.

However, the high-precision data from GEDI is difficult to read, analyze and visualize. And what the client currently used datasets processing tools: Google Earth Engine, which is a web-based programming tool to visualize the GEDI LIDAR data. But it requires JavaScript to explore datasets and like another web-based tool, it can not be used when it is offline As a result they did not give their required toolset they need to analyze the data easily and are easy to use online or offline.

By analyzing difficult parts for clients and what their current workflow is, a broad list of project requirements has been synthesized.

- The mobile application will display a map that can be navigated
    - Users are able to Scrolling and zooming color-coded maps
    - Users can select map types

- The mobile application will be capable of offline functionality
    - Users can download the region of interest for offline use

- The mobile application will display a map that can be navigated
    - Users can filter the map by input values
    - Users can get data values for a selected point

- The mobile application can support a multi-language interface
    - User can switch between French and English interface

- The mobile application will be capable of centering its map at a designated point
    - User can set default center location for next time use

By working hard for achieving all domain requirements, the team has created and completed the Android mobile application: Biomapper. Biomapper has solved the problem that the client had and met all the requirements. The Biomapper mobile application will process the dataset that our client gave to us and provide ecologists with an easy-to-use tool for research. Furthermore, the mobile application allows for ease of use both online and offline. As a result, it will give ecologists access to data

wherever they go and will best help our clients to deal with data without using Google Earth Engine or other programming languages. With this updated workflow, ecologists can spend more time collecting observations in the field.

Architecture and Implementation

Architectural Overview

This section provides an overview of the Biomapper application. It contains which modules are key to the operation of Biomapper. Specifically, the Android module and server module are the primary keys. The Android module is essentially the user interface. This is the primary way the users interact with Biomapper. The map tiles that are present when Biomapper opens is loaded from the server module. The server module which is handled on an AWS EC2 instance is where all of the required processing takes place for Biomapper operations. Such processing includes housing the Python script that is responsible for creating filtered map tiles.It also houses the Node/Express API that is responsible for responding to the many map tile requests from the Android module. Specifically, the API provides the logic of what map tiles are needed (filtered or unfiltered) depending on the condition toggled in the Android Module. Below is a diagram depicting the system design and reasons why the team selected each of the components.

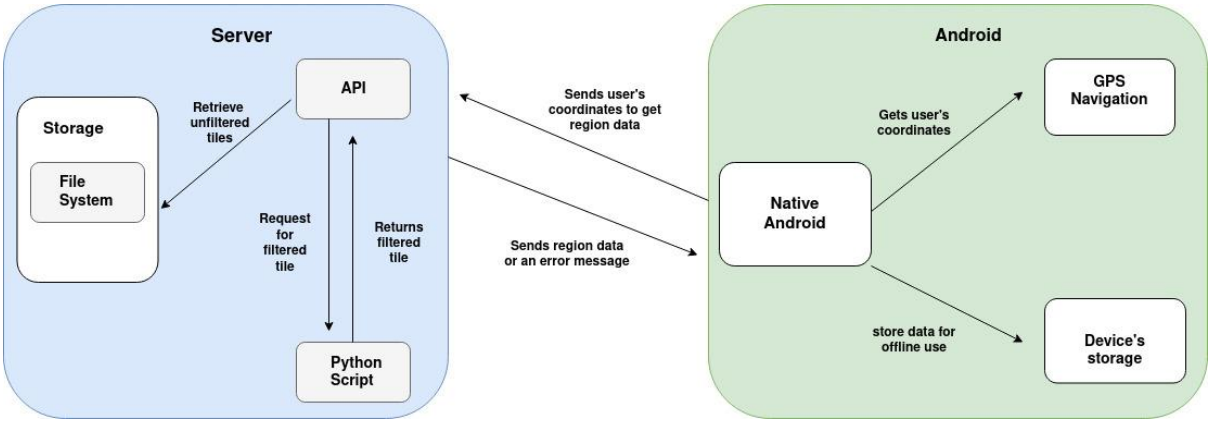


Figure 1: Biomapper Architectural Overview Diagram

Server Components

Node/Express API

The Node/Express API is a web service that allows communication between a user interface and back end component. For Biomapper, it allows communication between the Android module and server module. To go a little more in detail it listens for map tile requests from the Android module and responds accordingly.

Python Script

The Python Script is a source file written in the Python programming language that's primary task is to perform image processing on an unfiltered map tile..

File System/EC2 Instance

The file system on the EC2 instance is a standard Linux file system. The file system is not only responsible for hosting the API and Python script, but also unfiltered map tiles.

Android Components

Native Android Interface

The native Android interface is the core component of the Android module and is what the user interacts with. The interface is responsible for displaying a map of various African countries with several data types. It also allows the user to filter and toggle certain conditions for how they want the map to be displayed.

GPS Navigation

The Android module is entirely dependent on the device's navigation. The navigation pinpoints the user's location and displays the relevant data. When the user opens the application it defaults to the user's current location. These coordinates also provide a means of querying to the server module.

Device Storage

The device's storage is needed, for when the user wants to go into offline mode due to not having a network connection or whatever their situation may be. The user has the ability to input what data type and the metrics they want saved to their device.

Influences

The Biomapper application follows closely the client-server design paradigm. The paradigm has a server component that provides content to the client(s), and client(s)requests data from the server to view. This paradigm best fit Biomapper since the Android module needed to dynamically receive map tiles based on user location. Also, the paradigm strongly encourages that all processing is done on server-side rather than client-side to preserve client resources.

Each of the components in Biomapper's architecture has been chosen through intensive research and comparison. There were other alternatives, but the team felt the chosen components were the best to make the best product. The following paragraphs outline the rationale for choosing the components.

Native Android was chosen as the primary user interface for a few reasons. One being that the client's primary devices are Android and the team wanted it to be one-hundred percent compatible with their daily devices. The second reason is that developing a native Android codebase gives massive performance boosts over developing a cross-platform codebase such as React Native and Flutter. The team wanted Biomapper to perform its best given the hardware it would be running on.

The AWS EC2 instance was chosen because the server needed to be hosted on a cloud computing service for maximum performance. Also, the clients currently have their other projects hosted on AWS. The amount of processing that the server does needs to be hosted on a high performance platform to make sure there aren't any bottlenecks. Also, AWS EC2 requires a private key to access it, the team wanted the server to be secured as much as possible.

Node/Express API was chosen for a few factors. The main one being the simplicity of developing Node/Express code. It follows a minimal philosophy: it doesn't include external packages, this makes it so the developer decides what tools they need. The other reason is because of how simple it is to develop it in, it made it intuitive in how the Android module communicates with the server. The last reason is there are a ton of online resources for Node/Express. This was important for the team as we wanted to somewhat future proof Biomapper. In the sense that if something does go wrong and something needs to be rewritten there is a resource online that outlines what the issue is and how to fix it.

Component Interactions

To provide a better understanding of how each of these components interact with each other, the Biomapper component interactions are described in further detail below. These interactions correspond to the Figure 1 diagram.

Node/Express API < - - - > Python Script

When a request is made to the server for a map tile. The API checks for two additional parameters "min" and "max". If those two additional parameters are given, the API then knows that the user wants a filtered tile and sends the appropriate information to the Python script to filter the unfiltered map tile. The Python Script then returns the tile back to the API to return to the Android module.

Node/Express API - - - > File System

Unfiltered tiles by default are on the file system. The API then needs to query the file system for the map tile that was requested by the Android module. Once the tile is found the API simply returns the unfiltered tile.

Native Android - - - > GPS Navigation

As stated earlier, Biomapper is completely dependent on the user's location. The location helps Biomapper query the server to get relevant data for the user's location. If the user needs to get information that is not close to them, this is also possible through the use of GPS navigation.

Native Android - - -> Device's storage
When the user goes into offline mode, they will need to have the data saved to their device's storage if they want to continue their use of Biomapper.

Node/Express API < - - - > Native Android
This interaction can be argued as the most important interaction since this is what causes Biomapper to function correctly. The Android side sends the user's coordinates as parameters to the server in order to receive either an unfiltered or filtered map tile. The API is constantly listening for requests and either responds back with what the user needed or an error message.

Implementation Overview

Android

As mentioned before in the process overview part, Biomapper is an Android application that allows users to view different types of geographic data and has interactions with the application to make it more efficient to display. The team designed and implemented the Android application's main functionalities as listed below:

Display Geographic Map
Dynamically displaying different types of geographic maps is the main functionality to support the application work and it has the highest priority over other functionalities. To display the map offline or online, the team was developing and implemented the HTTP request and API request to get the target type of map data tiles from the server. More, the application has stable interactions and connections with the server that can support the application and display the target map smoothly and completely. By using those requests that the application will use the Google Map methods to provide the underlying map and display the geographic map on the application interface.

Multi-Language
By considering the client has their cooperation group in central Africa who has French as the first language and designed for more convenience to our client to utilize the application. The team was designing the application with multilanguage functionality. Compare with the locale language switch which was fundamental support by the Android Studio Java package, our functionality is able to switch the language by taking place strings that can be set for any other language. With multi-language functionality completed designed and implemented, the client is able to switch the language and also be able to modify the interface string which they preferred. As a result, it will give the client significant flexibility in language selection.

Offline Mode
By considering the client's workflow, the team designed and implemented an offline mode for the client and their research group while they are in a non-signal area. Offline

mode was designed in two parts, the ROI set, and the Download manager. The ROI part will set the area that the user was interested in then the application will use the algorithm to calculate the radius for the best fit for the area. Then the download manager will download it to internal storage in an asynchronous step. While the offline mode is on, the application will load the downloaded area and display them on the Biomapper application interface. The offline mode will significantly improve the client workflow and make them accessible to map data in a tropical forest.

Server

As mentioned earlier on, the server is hosted on an AWS EC2 instance. What lives on this instance is the filesystem that contains unfiltered map tiles, Node/Express API, and Python script.

The API which the server uses to communicate with the Android module is developed in Node.JS and Express.JS. Node.JS provides a runtime environment that allows JavaScript code to be executed outside of the browser. This means Node.JS is absolutely required in order to use Express.JS. Express.JS does the workload of the server as it is responsible for routing to respond to the different requests from the Android module. Specifically the routing is based on specified urls, so the API knows exactly how to perform. The code defined in those routes are consistent across the three datatypes of AGB, CHM, and DEM in terms of how it was written.

Final Thoughts

Designing and implementing the Biomapper's front-end part has flexibility and improvement with more communication with the client, the team was designing the application based on requirement documents and adding, revising, and improving the application when the application gets updated. By demonstrating to the client more updated functionalities, the detail and desired features were completed successfully as the client expected.

The overall implementation was similar to the architecture but there were a couple differences. The main one being that on the server the team originally wanted a concrete error handling system. However, this design idea was abandoned as the server functionalities are quite simplistic and didn't need a full fledged error handling module. The current error handling is handled in a few simple condition statements, which proved to be more than sufficient. The team acknowledges that the server may require additional functionalities if the clients' requirements change. It would be recommended that future developers design a sort of error handling in parallel to the additional functionalities.

Testing

To verify that the architecture in the previous section is implemented correctly, the team used a three component testing system. The first component was unit testing, this was to ensure that each individual unit of the application worked as expected. The second component was integration testing, this kind of testing checked to see if the mobile application and server modules worked together. The final component was usability testing, this consisted of testing the application among the clients' research team for feedback on simplicity, ease of navigation, and usefulness.

Unit Testing

The team has looked at specific sections of both the server module and Android module. The main functionality that was tested on the server was the Python filtering script. The Python script is what colors the different values of data. There were several tests of functionality conducted on the Android side, as it is more extensive in comparison to the server. The primary focus here was on the components that provided functionality. Components such as the AboutPage, Preferences, and ActionMenu were not tested.

The team decided to use JUnit for unit testing the Android application, it is a unit testing framework written in the Java programming language. It is appropriate since the app is written in the Java programming language and relies on the Gradle build automation tool, both of which JUnit is designed for. The framework was used to test **select** methods in the BaseMap file such as getBaseTileUrlString, getFilteredTileUrlString, RoiMarker, and setFilterValues to show validation of their accuracy.

Server unit testing was carried out using the Jasmine framework and a tool called JS-ImageDiff. Built on JavaScript, these are compatible with the team's Node JS.API. It automated testing the team needed to prove data integrity. JS-ImageDiff is specific to the primary functionality of the server, which is providing images of map data to the Android app.

The unit testing validated that the individual units of the application did behave as expected. As an overview, the server did respond back with a map tile if it was a valid request, if the request was invalid the server responded with an error message. The utility functions that were tested in the Android application responded back with the correct format of URLs that will be sent to the server.

Integration Testing

In addition to unit testing, integration testing was another part of the testing strategy. Integration tests combine many different modules and functions in a group to do the test and find out if the interfaces between them are correctly implemented and respond

accordingly. Integration tests on a component-based front-end such as a mobile application can be tricky to test. It is vital that the individual components of the mobile application are tested before stacking them to make up the Android module. The testing focused on the following modules: map tile retrieval and filtering, online map viewing and filtering, and downloading filtered/unfiltered tiles based on RoI.

The first test tested for map tile retrieval and filtering, or in other words the communication interface between the Android application and the server. The server portion involved receiving a request from the application side and returning map tiles from the AWS EC2 instance where the Node API resides. This interaction was done through HTTP requests and API calls from the application to the server. The result from this was that the application was able to retrieve both filtered and unfiltered tiles from the server depending on which condition was toggled.

The other test was to check that the ROI module within the Android module worked accordingly. The ROI manager is responsible for pinpointing an area of interest to the user and being able to download map tiles in that area. This tested the interaction between the ROI manager (application) and the device's internal storage. This test resulted in map tiles being asynchronously downloaded to the device's storage. This meant the user can then use the application in offline mode.

These tests tested the functionality between the application, server, and the device's storage. At this current time, there are no plans to add any additional modules.

Integration Testing

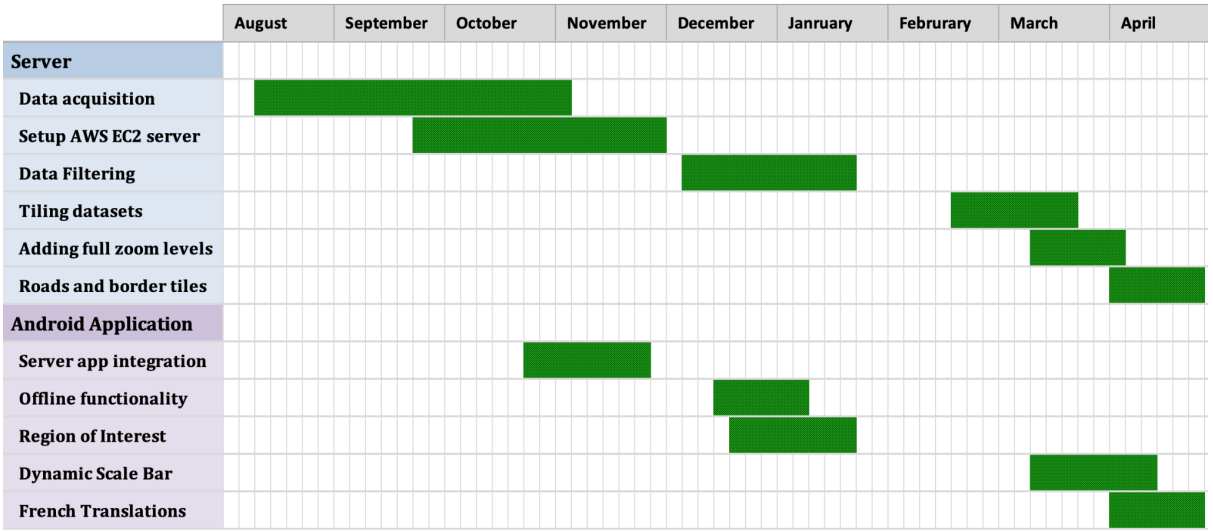
The last component of the testing system was usability testing. Its purpose was to test the Android interface to ensure that the functionality created was easily understandable and can be used by field ecologists with various degrees of mobile device knowledge.

This aspect of testing was conducted with the client's research team of eight researchers. The rationale was simply to have the people who would be the main users of the application to be the main testers of the application. That way the team could see how this would be used in a day to day operation.

The application was distributed to the team along with a chart to gather feedback on. The tests were performed during an in person meeting which the team wasn't present for. The team was given the feedback a few days afterwards.

To the team's surprise, the application was well received as the tasks that were listed on the feedback chart directly correlated to what they needed. The team was content with the feedback provided and didn't feel the need for major improvements and can continue to develop the remaining requirements.

Project Timeline



During the completion of our project there were many milestones that contributed towards the team's success. The hardest part came from data acquisition and took a large amount of time to research and apply new tools. The backbone of our application is an AWS EC2 instance which is setup to run a NodeJS API. With a custom written Python script, the API can take a user's request for a tile and return a filtered version. Once we had tested these key components for functionality, the tiling of datasets began. This was an unexpectedly difficult challenge due to the computational complexity of such large data sets. The server was completed once the roads and border tiles were added.

Looking back at the Android application there are a few major milestones that stand out. First was the filtering and downloading of tiles for offline use. Next was accessing data values for a given point from a downloaded tile. This feature allows the user to get an estimate of the data value at those coordinates. Last was integrating a dynamic scale bar which gives the user a visual representation of distance when viewing the map. Our team is putting the final touches on the Android application and uploading it to the Google play store soon.

Future Work

There are a few ideas our team has for future development of BioSphere. Starting with the development of an iOS application. This was originally included in the scope of the project but after researching cross-platform frameworks there were too many outdated or packages that had lost backing. Our team would have to redesign these larger packages to be able to use a cross-platform framework such as React Native. There are many benefits to choosing a native application such as performance and device integration. However, the downside to only choosing Android is that we limit our users to only this platform.

Thinking big picture when Biomapper proves to be beneficial to researchers then expansions of regions supported would be a great idea. Specifically, inclusion of South America which contains the largest tropical rainforest on Earth. This would allow access for researchers in those regions. To offset higher costs in storage and computing power, the app could be monetized but is up to the clients to decide. Limitations in storage size, computational complexity, and costs kept our team from pursuing this on our own.

The last improvement we suggest is upgrading the servers performance. Currently, a large amount of time is spent tiling and uploading data to the server. The performance of a high end personal computer is often not enough to quickly perform this task. Either having access to a supercomputer to reduce monthly costs or upgrading the server itself are good options.

Conclusion

As Figure 3 shows, the raw LIDAR data obtained from satellites must be processed and visualized so that others can make use of it. This is difficult to do as forest ecologists want the data in a simple format like a color-coded map, but the raw LIDAR data captures the environment in 3D. The waveforms that LIDAR sensors create to depict the 3D structure of the planet are complex and not readily viewable. They carry a number of datasets, including the previously mentioned CHMs, DEMs, and AGB. Programming scripts must be written that take this information and turn it into formats that are more human-readable, such as images. This undertaking requires technical knowledge that typical ecologists are unlikely to possess.

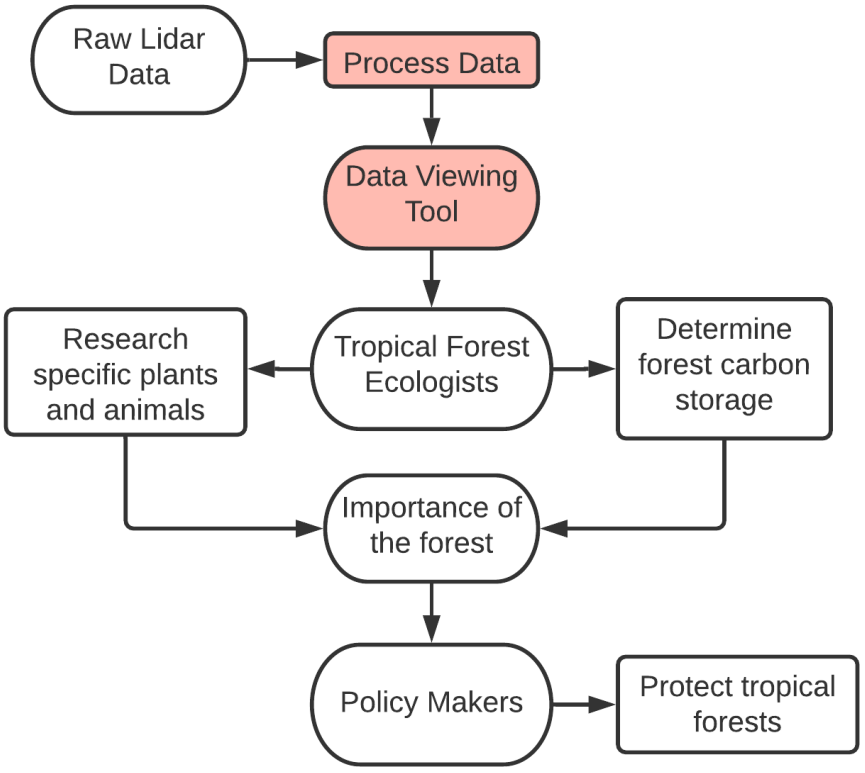


Figure 3: Problem Diagram

Luckily, the clients (as well as others) are capable of processing the raw data and making it available for others to use. They handle the difficult task of producing the datasets that are desperately needed to study, draw conclusions about, and protect tropical forests. This left the team with the task of creating a tool capable of displaying those datasets. The existing methods such as GEE or simply viewing data images are not sufficient. Ecologists need something that is more practical than GEE and provides more functionality than plain images. Ideally, it would be a tool they can take with them while conducting fieldwork.

The solution is a mobile application called Biomapper that is capable of displaying GEDI-derived data on a map. Biomapper will primarily provide data such as canopy height, digital elevation model values, and above-ground-biomass through an interactive map. The main features of Biomapper include:

- Ability to scroll and zoom in on the map, allowing the user to view any region of interest they desire.
- Ability to select the type of data to be displayed on the map.
- Ability to specify a range of values; highlighting areas within that range of input values.
- Ability to download maps for offline use.

With Biomapper, team Biosphere aims to help the clients reduce time by providing an easy to use tool for dealing with GEDI-derived data. It should help them spend more time worrying about the environment and less about the nitty gritty of making user friendly datasets.

The capstone experience has taught us that the keys to a successful project are patience, collaboration, and communication. Our team has experienced and overcame many frustrations and complications during the development of Biomapper. However, these experiences have given us the knowledge that we will use in real-world problems. It has taught us to adapt quickly and efficiently as possible.

Glossary

Above Ground Biomass(ABG): Combined biomass of above ground plant components.

Application Programming Interface (API): A software engineering mechanism that enables communication between software components using a set of definitions and protocols.

AWS EC2: A component of Amazon Web Services that allows users to rent virtual computers on which to run their own computer applications.

Digital Elevation Model (DEM): A graphical representation of elevation data to represent terrain.

Discord: A platform that allows users to communicate via instant messaging, voice calls, video calls, and other communication mediums.

ExpressJS: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Global Ecosystem Dynamics Investigation (GEDI): High resolution laser ranging of Earth's forests and topography from the international Space Station (ISS).

Google Earth Engine: A geospatial processing service.

Integration Testing: A form of testing that involves testing various modules as a group to validate functionality between them.

Jasmine: An open-source testing framework for JavaScript.

JUnit: A unit testing framework for the Java programming language.

Node.js: A backend JavaScript runtime environment that executes JavaScript code outside of a web browser.

Unit Testing: A software engineering technique that involves testing an individual unit or component of a software application.

Usability testing: Testing an application with a target audience to measure ease of use and user-friendliness of the software application.

Trello: A platform that follows a Kanban methodology for listing and assigning tasks.

Appendix A: Development Environment and Toolchain

Hardware

As a team, we used different systems to run and build Biomapper. In the following table, some specifications are provided to show what each team member used.

Matt	Windows / Intel processor (i7-8705G) / 16 GB RAM *No major problems. I found that our tiling process was easier on Linux however, so I made use of a Linux Mint Xfce virtual machine.*
Teng	macOS / Apple M1 / 16 GB RAM *To use macOS to develop applications is smoothly and successfully, the debugger and emulator for Android Studio have fast response speeds for the front end and back end.*
Brandon	macOS / Intel processor (i5) / 16 GB RAM *There were no problems encountered due to the operating system. When working with larger data the hardware limitations of my computer started to show.*
D'Yanni	Linux / Intel processor (i5) / 8 GB RAM *Linux didn't provide any obstacles, as the required software was Linux friendly. 8 GB of memory was sufficient enough to run Biomapper.*

Toolchain

The team used a variety of tools to develop, build, and deploy Biomapper. In the following table are a list of tools that each team member used. Some of these are purely personal preference and may be substituted out. However, the ones marked with (*) are absolutely required.

Matt	*Android Studio Google Earth Engine VirtualBox *GDAL IDLE (Python)
Teng	*Android Studio(Java/Kotlin) Parallel Desktop(Visual Machine) Google Earth Engine Netbeans(Java) PyCharm(Python)

Brandon	Microsoft Visual Studio Code GDAL *Node Package Manager (NPM) *Android Studio PyCharm (Python)
D'Yanni	Microsoft Visual Studio Code *Node Package Manager (NPM) *Android Studio

Setup and Production Cycle

For additional information about the necessary installations and maintenance for Biomapper, refer to the Biomapper User Manual. For information about how to use the Android application itself, refer to the Biomapper User Guide.
(https://biomapper.github.io)

To summarize the User Manual:

An AWS EC2 instance acts as the back-end server. It must have the following installed:

- HTTPD
- Node JS
- Python

The steps to creating the server are:

1. Create an EC2 instance
2. Install the technologies listed above
3. Run the Node JS application
4. Transfer the dataset tiles to the server
5. Modify the mobile app to the use the new server's IP address

For working on the modules of Biomapper, you'll need:

- An IDE for developing an Android app (like Android Studio)
- An IDE for developing the Node JS app (like Visual Studio Code)
- An IDE for developing Python script (like IDLE)
- Google Earth Engine for easily visualizing, modifying, and exporting datasets
- Software for creating map tiles (like GDAL, QGIS, MapTiler)
- A machine used to create the tiles (like a PC, virtual machine, cloud server, or supercomputer like NAU's Monsoon)

The steps to modifying the Android app are:

1. Install an Android IDE
2. Modify the codebase
3. Push changes to the Biomapper Android GitHub
4. Release an update for the app on the Google Play Store

The steps to tile a dataset are:

1. If the GeoTIFF dataset is not already in Google Earth Engine, add it as an asset. Add the new dataset's file path to our code base.
2. Export the dataset to your Google Drive and transfer to whichever computer will be performing the tiling.
3. If multiple files are produced, use GDAL to merge them into 1 virtual tile.
4. Run GDAL's tiling software for the desired zoom levels.
5. Export data to server if run on a separate machine using "tar" command