Software Design Document

01-24-2022

Version 2.0

Team BioSphere

Project Sponsors: Jenna M. Keany, Dr. Christopher Doughty

Team Mentor: Melissa D. Rose

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

Table of Contents

1. Introduction - p. 2

2. Implementation Overview - p. 5

3. Architectural Overview - p. 7

4. Module and Interface Descriptions - p. 9

5. Implementation Plan - p. 15

6. Conclusion - p. 16

1.0 Introduction

1.1 Background

The loss of tropical forests is a cause for global concern given that they play a vital role in large-scale environmental processes. By absorbing vast quantities of CO2, tropical forests provide oxygen and help stabilize the Earth's climate. Furthermore, tropical forests help to maintain the world's water cycle through transpiration, generating clouds that travel all over the world. In terms of biodiversity, "tropical forests contain over 30 million species of plants and animals... half of Earth's wildlife and at least two-thirds of its plant species!" [1]. These facts make it clear that tropical forests need to be preserved and protected.

The importance of environmental protection cannot be overstated. It is crucial that forest ecologists have access to and are able to interpret LIDAR-derived data products such as digital elevation models (DEM) and canopy height models (CHM). DEMs are a representation of the bare ground topographic surface model of the Earth and CHMs are a measurement of the height of trees, constructions, and other structures above the ground topography. DEM and CHM values are important for determining where specific species of plants and animals may be found. The results of these studies, along with values of above-ground biomass (AGB), help determine the environmental importance of forest ecosystems. AGB data provides an estimate of the carbon content of each area within a forest. Additionally, conclusions drawn by the ecologists allow policymakers and conservationists to better understand environmental changes and enact legislation to protect forests.

The aforementioned datasets can be obtained from LIDAR-equipped satellites, such as the International Space Station with its onboard GEDI sensor. However, processing this data and visualizing it is a non-trivial task. What forest ecologists need is an easy way to access the processed environmental data in real-time while conducting their fieldwork. Such a tool is what Dr. Christopher Doughty, the client, and sponsor of this project, has tasked the team to create. He studies megafauna within the forests of Africa and works closely with ecologists working there. The motivation of his lab's work is focused on learning about the environment and ensuring that it is protected. Currently, the geospatial data they use can be visualized using tools like Google Earth Engine (GEE). GEE is a web application that uses Javascript to host, edit, analyze, and visualize processes of geospatial data within a web browser. However, it has its unique codebase language that is not applicable to every user

and is difficult to use in the field. Thus, ecologists are still in need of a simple tool for visualizing analysis-ready environmental data and relevant maps.

1.2 Solution Vision

As was stated, forest ecologists are in need of an easy way to access processed environmental data in real-time while conducting their fieldwork. This will be achieved through a mobile application called Biomapper. This app is designed for users to be able to access relevant map data whether online or offline. The interactive map contains the raster data of DEMs, CHMs, and AGBs models in several African countries and it can also be modified to choose what kind of layer that the user would like to access. Based on the map data type, the application server will filter the target layer for the user and select by user's selection to show. It can contain lots of different types including DEMs, CHMs, and AGBs. And more layers could be added in the future.

Design planning of this mobile application consists of three parts: map data acquisition, server development, and Android application development. The detail for these three parts will be introduced in the Architectural Overview section and Interface Description sections of this document. The details will be introduced in the following sections, in a simple way to introduce the application that is a mobile phone application which will come up with Android Edition and iOS Edition, and it will contain functionality that can change the user mode to online or offline. In online mode, it can dynamically access the data through the server and can execute functionalities such as data filtering, manual location selection, and language selection between English and French. As far as the data filtering/ location selection will be done, the selected map by the user can be shown on the Main Map interface. Moreover, the layer and area which was selected by the user can be downloaded to the mobile phone's storage and it could be accessed offline. This functionality is for the researchers who were in African areas without networking.

The main functionality of this application is to allow users to select different layers (CHMs, DEMs, AGMs) in middle Africa (Gabon, Congo, and all possible countries that are needed), and to provide visualization of the data. Specifically, it will have the following features and functionalities:

- The application is able to access the new map tile from the GEE script
- The application can be used in offline mode
- Users can choose an area and download it for offline use
- Users can choose between different layers in the application
- Users can search for a given data range and visualize the results on the map
- Users can click on a location and get data for that point

- Users can choose the language (English, French)
- Users can set the default center location

With these functionalities, the final product will provide users with a map application that allows them to utilize satellite LIDAR data efficiently.

2.0 Implementation Overview

The proposed solution is to create a mobile application called Biomapper that is capable of delivering reliable GEDI-derived data to on-site researchers. The application will primarily provide data such as CHM, DEM, and AGB values through an interactive heat map.

The data that is going to be needed will be large in quantity and therefore will need to be hosted somewhere that supports extensive storage. The solution to this is to use a cloud server such as AWS. The team has decided to utilize AWS's EC2 service. What EC2 provides is a Linux based server that supports ease of use such as installing backend components like Node.js. EC2 being a Linux server also enables ease of use when installing other packages that might be needed during server development.

The data that will be collected from GEE will be sent to the server's file system. Also on this server is the Node.js backend, which will be the intermediary between the mobile application and the GEDI-derived data. The purpose of this backend is to actively listen to requests from the mobile application and respond accordingly with the right data tile. The backend will also have error handling that will respond to faulty requests with error messages. It will do this by using a few conditional statements in controller code that checks if the tile that is being requested does in fact exist on the server. If the tile does exist then that data tile is sent to the mobile application. Otherwise, an error message or in this case a red tile will be sent back since the mobile application is expecting tiles from the server.

As the belief that a mobile device will serve best as the tool for on site researchers, this leads into the next core component, utilizing a native Android application. The application is going to be sending tile requests anytime the user is scrolling on the map. Android abstracts this action by having a url function that acts as a GET request for the application. These HTTP requests are going to contain metadata such as the data type, zoom level, x coordinate, and y coordinate. The server is going to listen for the HTTP request and respond with a data tile. The application is going to take the data tiles and display them accordingly.

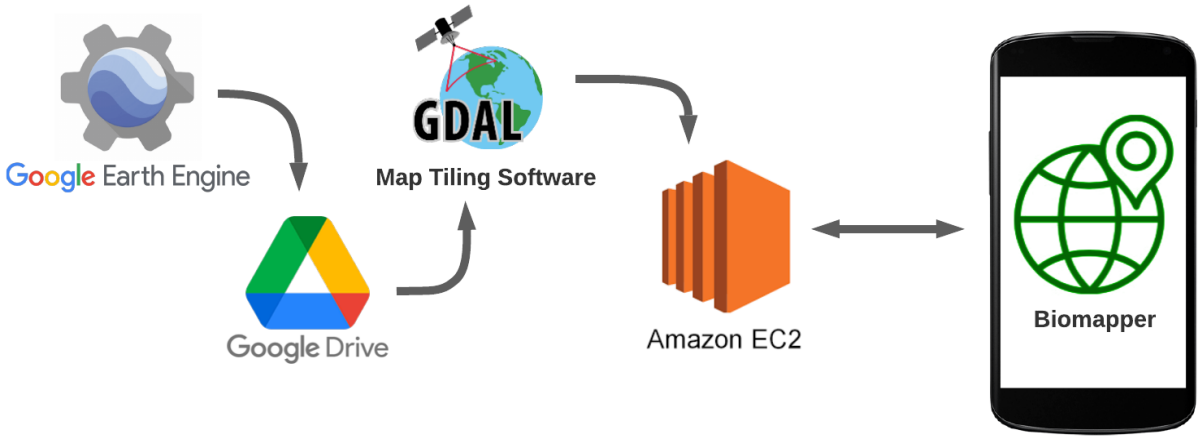

Figure 1: Map Tile Creation and Retrieval

3.0 Architectural Overview

3.1 Data Acquisition

The data required for the Android application must first be obtained from where it is currently stored. Google Earth Engine (GEE) is a platform for storage, visualization, and analysis of geospatial datasets. A script will be created within GEE that exports the needed data as images to a Google Drive. Then, map tiling software will be used to split these images into many map tiles stored in a nested folder structure. This is necessary since the map rendering software used within the mobile application, Google Maps, relies on map tiles.

3.2 Server

The backend server will store map tiles and provide an API that the mobile application requires to function. This API allows the mobile application to retrieve map tiles via HTTP requests. This includes the ability to get modified map tiles based on a range of data values provided by the app user. The API will also feature an error management service that ensures only properly formatted requests are handled and given the appropriate responses. This software server is run on an Amazon Web Services EC2 instance, which is a virtual machine that represents a physical server.

3.3 Android Application

The Android application will consist of two primary components: the main map and the action menu. The main map allows the user to zoom and scroll the data maps so that they can find and view areas of interest. The action menu provides a list of possible actions and settings that the user can choose from. This list will be filled with options like selecting the desired type of data, setting the range of data values the user wants shown, choosing a location that is needed for offline use, etc.

Each of these components is implemented using components known as Fragments. These components are modular and thus make it possible for them to be reused. For example, it would be a waste of resources to recreate the map when the user wants to select a location for offline use from within the action menu. Instead, this option can use the same base map that was used in the main map. This is one of the many reasons why Fragments will be used heavily, instead of the alternative components known as Activities.

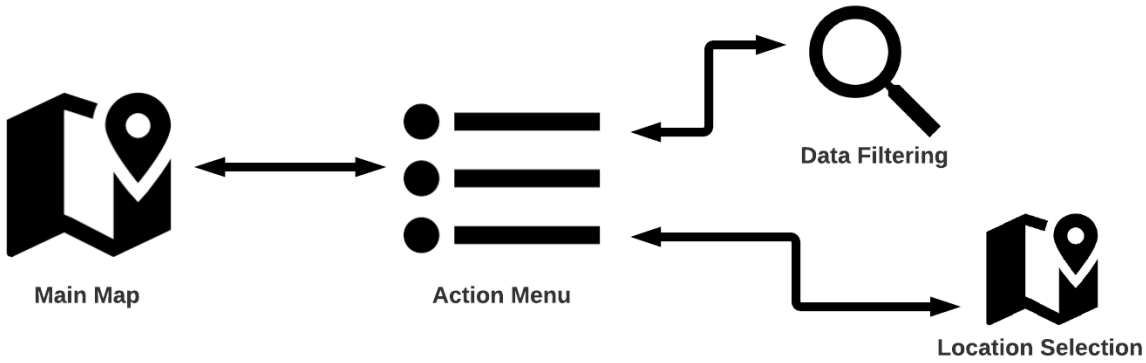

Figure 2: Mobile App Navigation

4.0 Module and Interface Descriptions

4.1 Data Acquisition

The data used by the mobile application must initially be retrieved from GEE. Then, the exported images must be processed using map tiling software before being transferred to the server.

Google Earth Engine

A script (also known in GEE as an 'app') will be created that imports datasets for CHM, DEM, and AGB. The script also defines the boundaries and color palettes that are applied to these datasets. Finally, it will allow for the exporting of the data as large GeoTIFF images. This image type provides georeferencing for each of the pixels of the image. The individual doing the exporting (and presumably adding map tiles to the server) will have these images added to their Google Drive. They will then download them to their machine before completing the next step.

Map Tiling

Then, map tiling software called **gdal2tiles** will be used that splits these images into many map tiles stored in a folder structure. The created folders are structured like so: "root/<zoom-level>/<x-coordinate>/<y-coordinate>.png", with the coordinates being based on the Spherical Mercator coordinate reference system (also known as EPSG: 3857). Because the images provided initially are GeoTIFFs, the software can determine which coordinates are appropriate for each of the map tiles. The tiling process is necessary since map rendering software like Google Maps rely on map tiles.

Transferring to Server

Finally, the folders of map tiles will be transferred from the machine where the tiling was done to the server. This is done by using the 'scp' command, which allows one to transfer files from their system to a different one. Then, one must SSH into the server to use the 'mv' command to move the map tiles to the appropriate location. This makes it possible for them to be sent in responses by the server to the mobile application.

4.2 Server

The following section describes the backend side of the Biomapper project. The backend contains a RESTful API that will be called Biomapper REST API. Biomapper REST API provides resources and endpoints to call the different services that connect the entire server infrastructure together. The services that will be provided by the backend are a Retrieve Tile Service (RTS) and an Error Management Service (EMS).

Retrieve Tile Service (RTS)

The Retrieve Tile Service (RTS) is an internal service that provides the mobile application the ability to retrieve data tiles depending on where the user is scrolling on the map. This service will get its input from the coordinates of the map to retrieve the correct tile. A diagram of the envisioned service is shown in Figure 3.

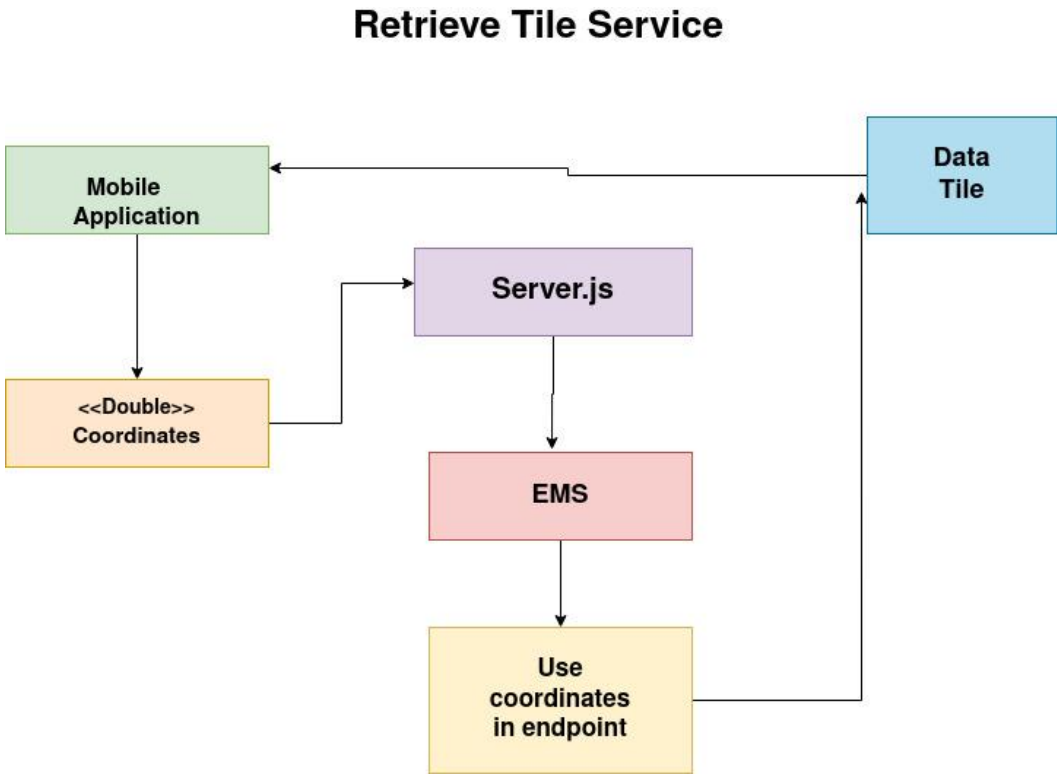

Figure 3: Retrieve Tile Service (RTS) Design Diagram

Figure 3 outlines how the Retrieve Tile Service (RTS) will get coordinates and return the corresponding data tile to the mobile application. This process starts at the mobile application level. When the user is scrolling through the map they are actually scrolling through coordinates. The coordinates are in the data type of an Integer (whole number) which means the server is expecting an integer in the HTTP request. Once it gets to the server the HTTP request will be sent to EMS (explained below). Now that the request has been sanitized and validated as a correct request it will be used as input in the endpoint to find the corresponding data tile. Once a match has been found the server will send this data tile as an HTTP response to the mobile application.

Error Management Service (EMS)

The Error Management Service is an internal service that provides error handling functionality to the server, this is meant as protection for the server. The server is expecting only a certain kind of request from the mobile application. To reiterate the server and the mobile application are developed independently. With that said the server does not know what kind of requests the mobile application will be sending to it. EMS comes in and verifies that the HTTP request is in fact a valid request that the server can understand before any "processing" can be done. A diagram of the envisioned service is shown in Figure 4.

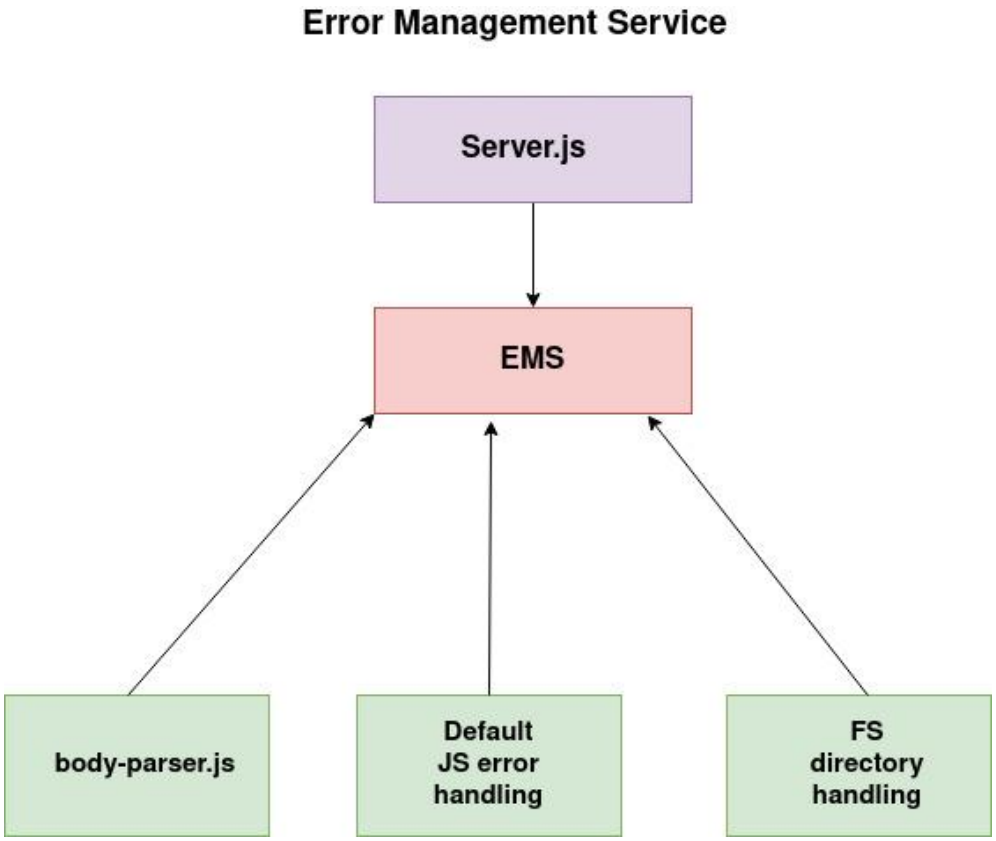

Figure 4: Error Management Service (EMS) Design Diagram

Figure 4 outlines how the Error Management Service (EMS) will take an incoming HTTP request and perform preprocessing before the server processes the request. By looking at the diagram it can be observed that EMS is actually composed of various services. The simplest service to start with is the one native JavaScript (JS) provides. By default JS offers basic error handling such as exceptions and try/catch statements. File System (FS) error handling is similar to native JS error handling, except it deals with file system structure. FS error handling takes place in the routing functions (where the server knows where to get data) and can deal with things such as a file not existing. Body-parser.js is a library that parses through HTTP requests and breaks it down by what it is. This is useful in the sense

that the server can stop here if something fails at this state. By design EMS will start with Body-parser to catch most errors, it will then have native JS/Express.js error handling afterwards to catch any errors that slipped by.

4.3 Android Application

This section depicts how the Android application will be designed. This includes the method used for creating the overall structure of the app, the programming languages used, and the specific classes created to accomplish the goals of the app.

Single-Activity Architecture

The app will be focused around a single-activity architecture. This means it will rely heavily on UI components called Fragments for functionality, rather than using multiple Activities. This is because Fragments require less resources, are generally easier to use, and can be reused as needed. However, Fragments are dependent on an Activity and cannot exist without them, so a single one must still be used.

Languages Used

As an Android application written within Android Studio, the primary programming language that will be used is Java. This provides the actual functionality of the app, and was chosen over the alternative language, Kotlin, because the team is more familiar with it. XML is also used heavily. It provides the layout that the app users see.

Classes Created

Each of the classes needed for the application are described below:

MainActivity extends AppCompatActivity

This class is the single activity used within the program. Since it starts the program and communicates data between the Fragments, all other classes are dependent on it. The only functionality it provides is starting the MainMap.

MainMap extends Fragment

The MainMap provides the primary view for seeing data on the map. This is done by including the layout of another Fragment called the BaseMap. Additionally, it has a floating action button that takes the user to the ActionMenu.

BaseMap extends Fragment

The BaseMap is responsible for displaying the map data to the user in multiple different scenarios. It provides all the map functionality such as scrolling and zooming, as well as loading in new map tiles.

ActionMenu extends Fragment
This class is the primary container for housing the Preferences list. It also adds a Toolbar with a back-button for navigating back to the MainMap.

Preferences extends PreferenceFragmentCompat
Provides a list of preferences or settings that the user can choose from. These subsequent options (and their respective classes) provide the majority of the app's functionality. The Preferences class is responsible for handling the opening of the other mentioned classes.

ApproveAction extends Fragment
This class is used when an action is taken that needs to be either approved or canceled by the user. This includes when setting a range of data filter values and when selecting an area on the map. The text within the Toolbar and functionality of the accept button will change depending on the scenario.

SetValueRange extends Fragment
Allows the user to input a minimum and a maximum value that are used for filtering the map data. The absolute maximum or minimum value displayed to the user is dependent on the current data type being shown on the map.

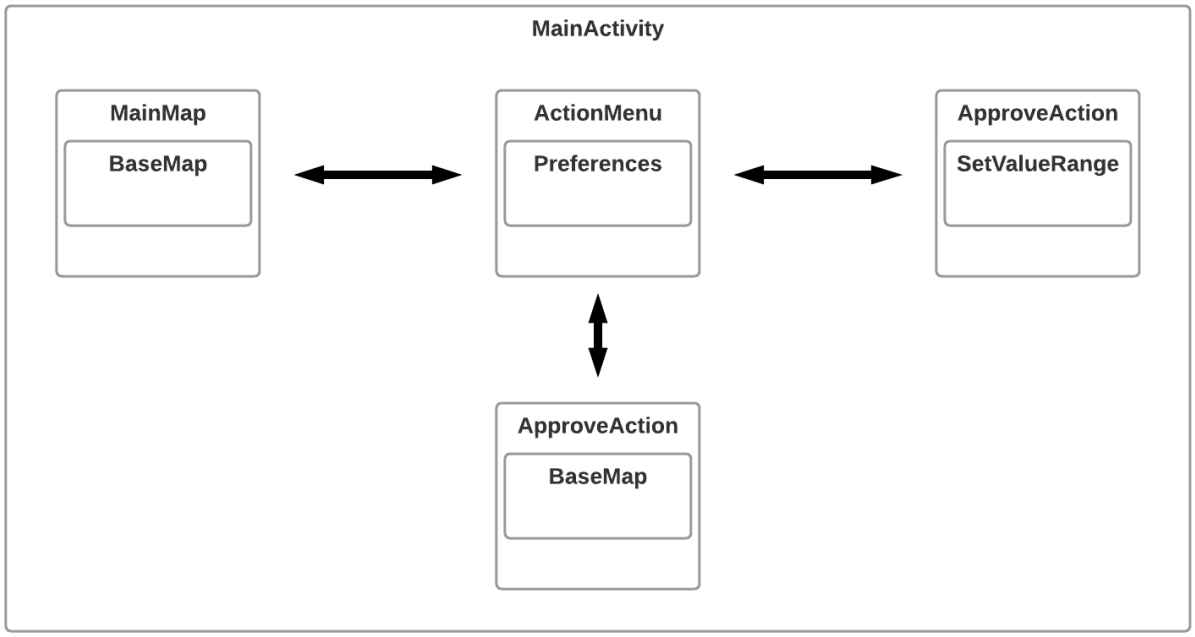

Figure 5: Mobile App Class Layout

Figure 6: Mobile App User Interface Layout

Implementation Plan

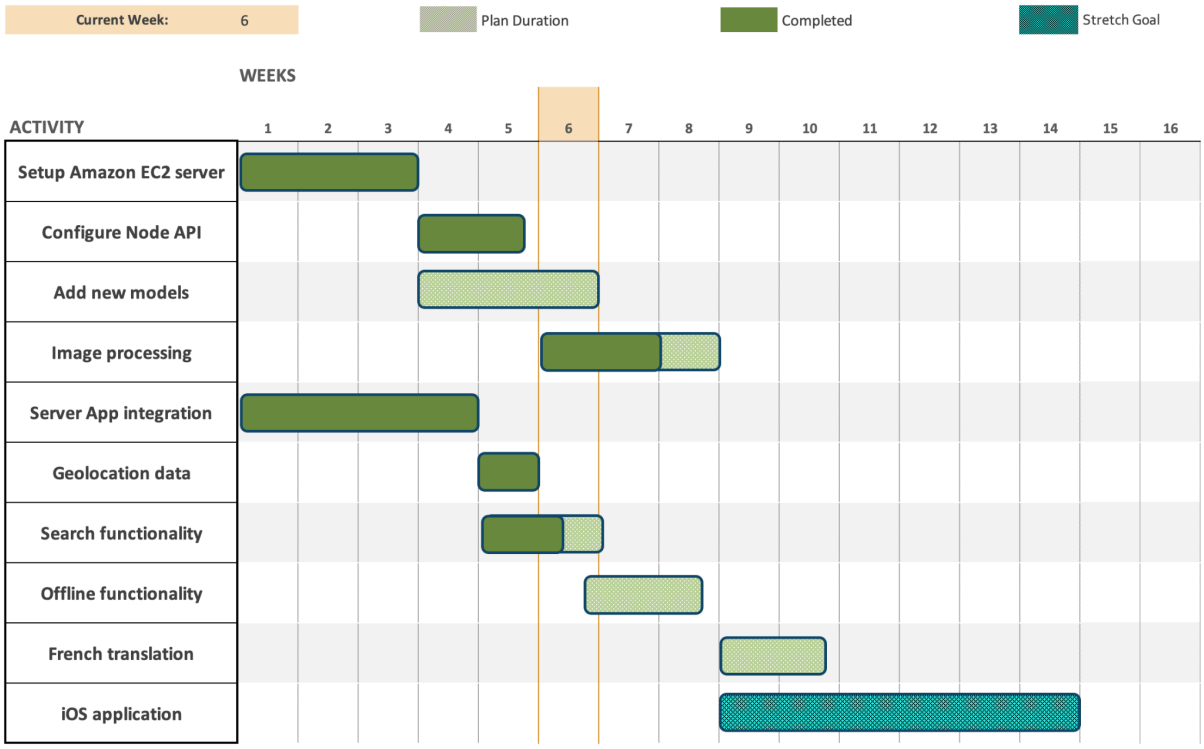

Figure 7: Gantt Chart Project Plan

Shown above is our team's schedule with various steps outlined for us to reach our goal on time. Funding for our server and storage has been secured making our next step will be combining the API and Image processing portions. This is crucial in developing a functional alpha demo by week 9. During this time, our team will be adding to CHM, DEM, and AGB models to improve quality and land coverage. These models will require our server storage size to be increased.

Development will continue the Android app with focus on creating the clients desired features. Recently our team completed accessing a user's geolocation data to center the map on their location. The next step is adding the search functionality which will filter the map tiles by sending requests to our server with user specified values. Following these steps is creating the offline functionality, this will require a selected region to be downloaded at a specified quality to the device. Lastly, once our application is mostly complete our team will be working with a French translator to add multilingual support. If development on the Android application is ahead of schedule, part of the team will start research and development on an iOS version. This would run concurrently with the finalizations of the Android application and the goal is to have it completed before the end of the semester.

Conclusion

The loss of tropical forests is cause for global concern; this is due to the vast amounts of CO2 absorbed by tropical forests. Crucial data that is derived from LIDAR imagery satellites such as digital elevation models and canopy height models can be used to interpret and provide insights into an ecosystem's health and biodiversity. The goal of this application is to provide the application and its background interpreter process to the team client and researchers who need the LIDAR data to analyze the African environmental data and draw the conclusion for the policymakers. And make the policy change as the data changed to improve the environment quality. And the application will have an optional download feature will be included to aid researchers who are conducting field studies where internet connection is not guaranteed. The application will include GPS data to center on a user's location, the ability to filter data to a specified range, and access data stored on a server to reduce space complexity. When the user interacts with this application, the goal is to provide an easy-to-use solution, intuitive and informative for all types of users.

This document laid out details for designing for the mobile application and a plan for reaching the client's needs. Coming to an agreement with the clients through this document will mean both parties are in accordance with what is expected moving forward. Indicating that team Biosphere has done their due diligence in research and communication to achieve the correct outcome. With progress on a prototype mobile application that has a navigable map and acquisition of data from GEE stored on a server, the team is making great progress and will continue development in the upcoming months.