Software Design Document

GeoKings - GeoSTAC

Sponsors - USGS Amy Stamile, USGS Trent Hare, USGS Jason Laura

Andrew Usvat, Zack Bryant, Alexander Poole,
Jackson Brittain, John Cardeccia

CS 486

Instructor: Dr. Leverington
Mentor: Vahid Nikoonejad Fard

9/21/2023

Northern Arizona University

Table of Contents

Introduction

The decades of space exploration by governmental agencies such as the National Aerospace & Aeronautics Administration (NASA), European Space Agency (ESA), Japan Aerospace Exploration Agency (JAXA), and others have resulted in volumes of data about the various planets of our solar system.

The amount of data, and the complexity and cost of the programs used to convert the raw data into Analysis Ready Data (ARD) make it difficult for the public to access. To solve this problem, the United States Geological Survey (USGS) created GeoSTAC, a web based map application - similar to Google maps, but for other planets - built by a series of NAU capstone teams. It provides users easy access to available ARD.

Problem Statement

The current implementation of GeoSTAC does an excellent job of delivering ARD to users. A large amount of available data, however, in the form of vector data (lines and polygons representing geologic features), cannot be accessed or displayed by GeoSTAC. Our clients have laid out the following requirements regarding the desired additional functionality of the application:

- Access the available vector data from a provided API.
- Customized styling of the vector data with icons and patterns.
- Implementation of user search functionality for the newly available data.

Along with these functional requirements there will also be some necessary performance requirements.

- Verifying that the correct data is being displayed and stylized.
- The UI changes necessary for users to be able to query the Vector API will need to be user friendly.
- Updating the existing application documentation to include our modifications.

All of this will need to be done within the constraints placed by our clients and prior capstone teams; a front-end written in ReactJS, rendering handled by Leaflet, and an ElasticSearch backend.

Successfully implementing the previously mentioned features will forge GeoSTAC into a more advanced, and even more useful data analysis tool for planetary scientists.

Solution

Our team proposes to integrate the new Vector API into the GeoSTAC application in a way that compliments the application without having to rebuild the entire existing project. The change our team makes will allow for the:

- Fetching and rendering the data via a fetch call, and rendering all vector data of points, lines, and polygons - stored in the GeoJSON format - with the JavaScript library Leaflet.
- Implementing and modifying the Leaflet.SLD package to display the correct icons and patterns, as described in the SLD files recommended by our client.

- Integrating ElasticSearch with the Vector API so that users are able to submit correctly formatted queries - per the Open Geospatial Consortium's (OGC) Common Query Language (CQL) specification - and have only the requested data returned by the API.

While integrating these features it will be necessary to implement testing to ensure the correct information is being rendered for the corresponding geologic feature. Additional testing with users will also be required to highlight any potential difficulty with the updated UI.

We see this implementation as an update to the overall system, expanding its capabilities and improving its usefulness.

Implementation Overview

Our client – the USGS – has an existing application that they wish to make improvements upon - their interplanetary mapping application GeoSTAC. Users will need to be able to view maps of not only the currently available data, but also view maps populated by vector data. The GeoKings team will modify the existing application so that it can also fetch data from the USGS Vector API and render the data using the existing Leaflet library.

The Leaflet library used by the prior capstone teams has an available package called Leaflet.SLD. We will implement a modified form of this package in order to apply stylization to the newly rendered vector data. The data used for stylization will come

from provided SVG files of icons and color patterns, developed by the geologic scientific community.

This new source of vector data will necessitate that users be able to sort through and select the data they wish to view. We will integrate ElasticSearch into the API calls, allowing user queries to return only the specified data from the Vector API. The ability to search through and return only the relevant data means that UI changes will need to be made. The existing UI – written in ReactJS – will be modified to include fields for users to select the desired data.

Architectural Overview

While the prior teams either created the base architecture, or heavily modified it, the necessary framework of the GeoSTAC application is now entirely in place. Instead of changing the existing architecture, our team will simply be expanding upon the existing framework with the addition of new functionality.

The diagram below (Figure 1) provides a visualization of what our added functionality will look like, and how the various functions and features will interact with one another. This will all largely (though not exclusively) take place within two files: AstroMap.js and Leaflet.SLD.

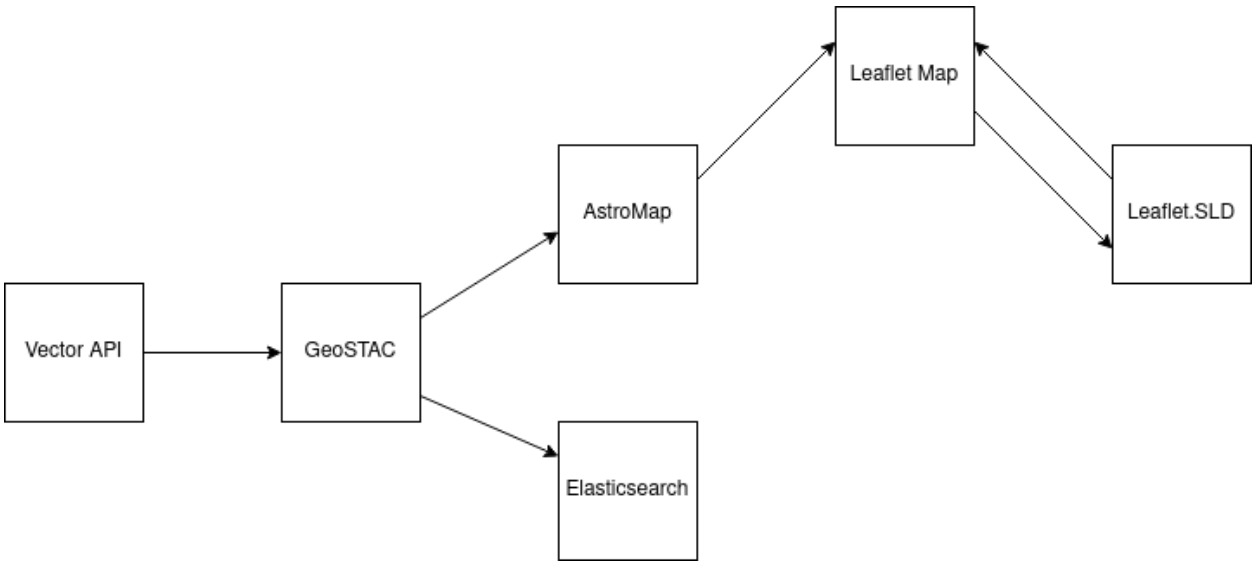

Figure 1: Diagram of Search, Vector API, AstroMap, and Leaflet interactions

The Vector API is the source of the vector data that we have been tasked with adding to the GeoSTAC application. Fetch commands will be run to pull the necessary data, passing it to GeoSTAC.

The search functionality being implemented here is handled by ElasticSearch - utilized by the request of our clients - and will allow for the filtering of the results for the specific geologic features desired by the user. It will be paired with a series of UI improvements to the front-end of the GeoSTAC application. The UI changes will be achieved by modifying the existing ReactJS code to allow users to interact with ElasticSearch and select from the available search parameters.

The data retrieved from the Vector API is passed to AstroMap, and then to Leaflet for rendering. From there, the Leaflet.SLD package will style it according to the features described in the data.

The final result will be a fully rendered map of vector data, with all of the selected geologic features being displayed and represented by the appropriate icons and patterns.

Module and Interface Descriptions

I. Stylizing

Leaflet.SLD is responsible for managing symbols on the Leaflet map for GeoJSON features. As of right now, Leaflet.SLD is being used in direct relation to AstroMap.js, this is because AstroMap.js holds the reasonability of rendering the GeoJSON features to the map and Leaflet.SLD is responsible for displaying these symbols onto these GeoJSON features. Leaflet.SLD will be called right after a feature is removed, refreshed or added to the Leaflet map in order to properly manage the visibility with respect to the GeoJSON features.

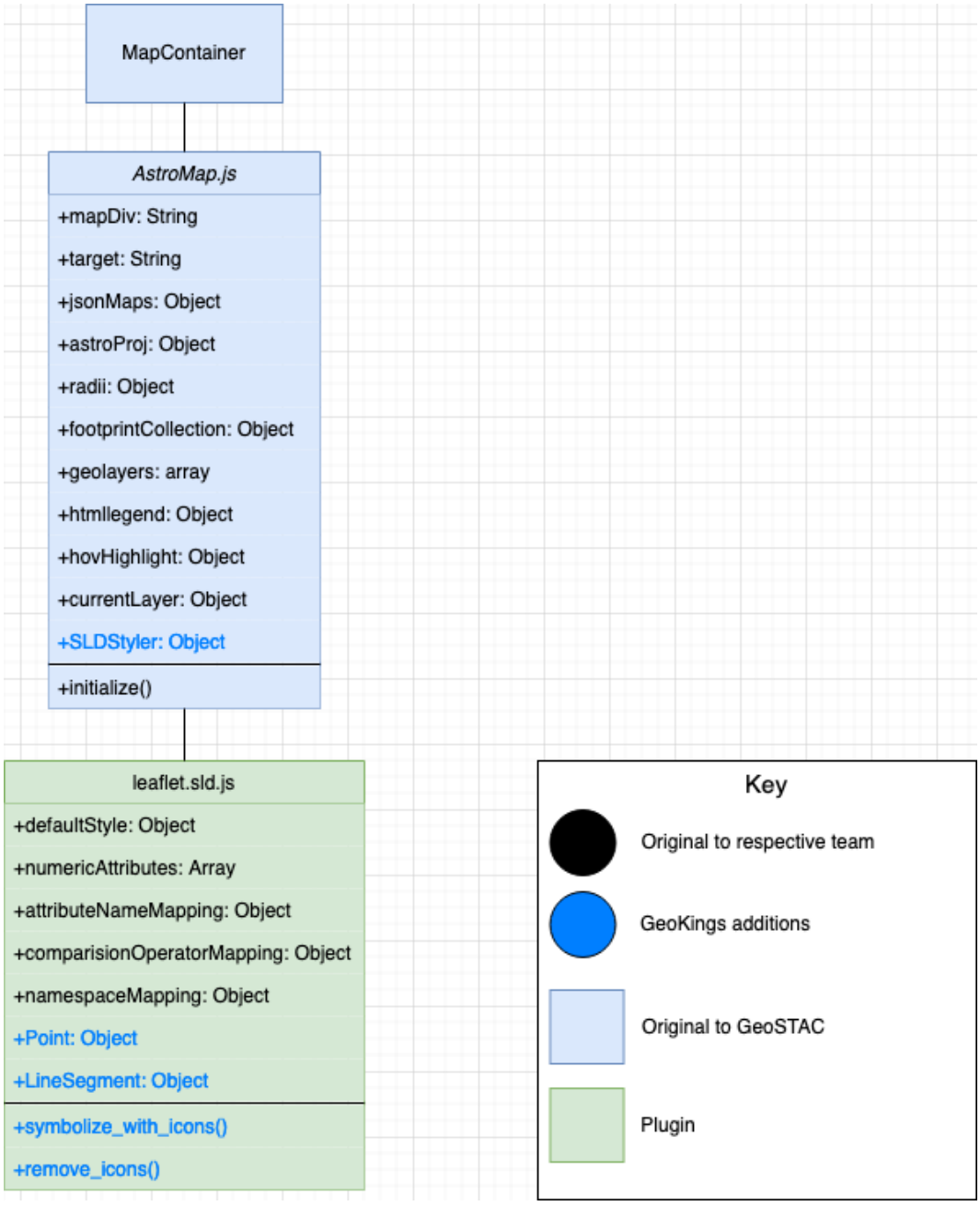

Figure 2: Leaflet.SLD and it's relations in GeoSTAC

Leaflet.sld has two publicly available methods. These methods are intended to be used solely for manipulating symbols on the Leaflet map. The methods add_symbol and remove_symbols will be used to either add symbols to the features visible on the map or remove those symbols no longer visible on the map respectively.

a. Symbolize_with_icons

Takes in a geolayer and a map object. From here the function will parse through the geolayer's features adding symbols to the map object. There is no return for this method.

b. Remove_symbol

Takes in a geolayer to be removed from the map. It is called after any time a geolayer is removed from the map. Remove_symbol will look at the latest geolayer that is removed and remove all symbols that were being displayed with that layer. It doesn't return anything as of right now.

II. API Fetch

The data implemented into our application is being fetched from 2 different API's. One of which, the SpatioTemporal Asset Catalog (STAC), was previously implemented by the GeoSTAC team. Now, the GeoKings have implemented an API containing vector style data points that differ heavily from the previous team. What this means is that we had to modify the previous works in order to fetch the new data correctly, and properly store, and send it to our React components for interaction. Below is a diagram showing the connection between the fetch data functionality with the FootprintFetcher React component.

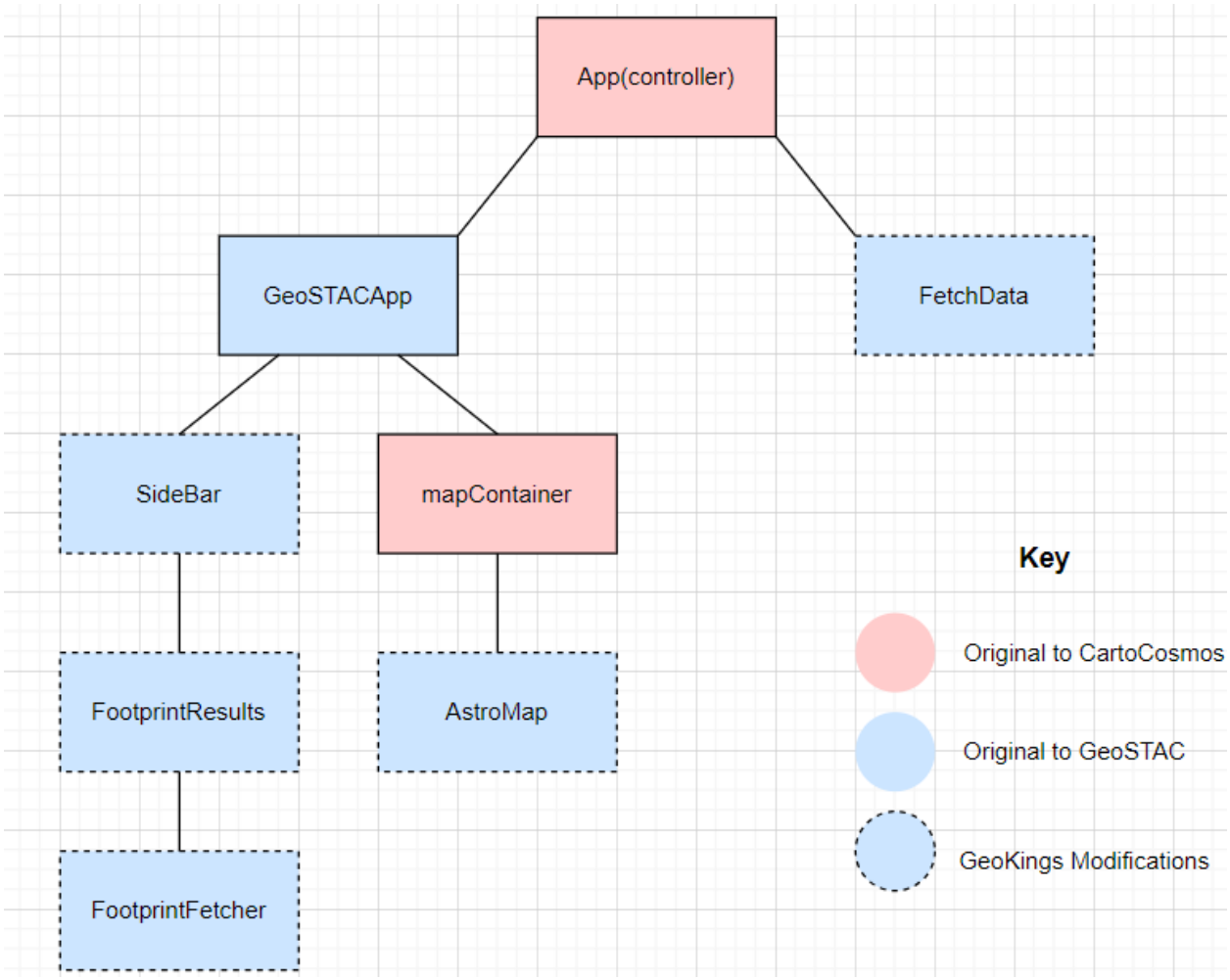

Figure 3: Diagram of API Fetch

a. FetchData

FetchData is the main component responsible for gathering the endpoints from two different API's. As of now, FetchData is handling both the Spatio Temporal Asset Catalogs (STAC) API in addition to the new vector API. The modifications we had to make here was to fetch the additional API to be used by the rest of the app and make sure that it can grab each data set the API contains.

b. FootprintFetcher

FootprintFetcher is a React component directly involved in fetching a specific number of footprints requested by the user. This first component is one that directly interacts with FetchData through its parent functions. Once the footprints are fetched, it then sends the data to FootprintResults to be sorted and displayed. With the modification of FetchData, we modified FootprintFetcher to decipher the data and to figure out which array to gather information from. Because we have multiple different data sets within the vector API, it is crucial that FootprintFetcher is able to understand where to grab that information from given a specific id.

III. Graphical User Interface (GUI)

The graphical interface of our GeoKings project serves a big purpose due to the fact that it requires interaction between the user and the interface. The major GUI components consist of displayed footprints, metadata that users can visualize and a sort/filter area for raster and the new addition of vector data. That being said, this requires modifying React components that were in the existing GeoSTAC structure. Our Figure 4 below provides a visual representation of the original and modified components for the GUI. These components are highlighted appropriately.

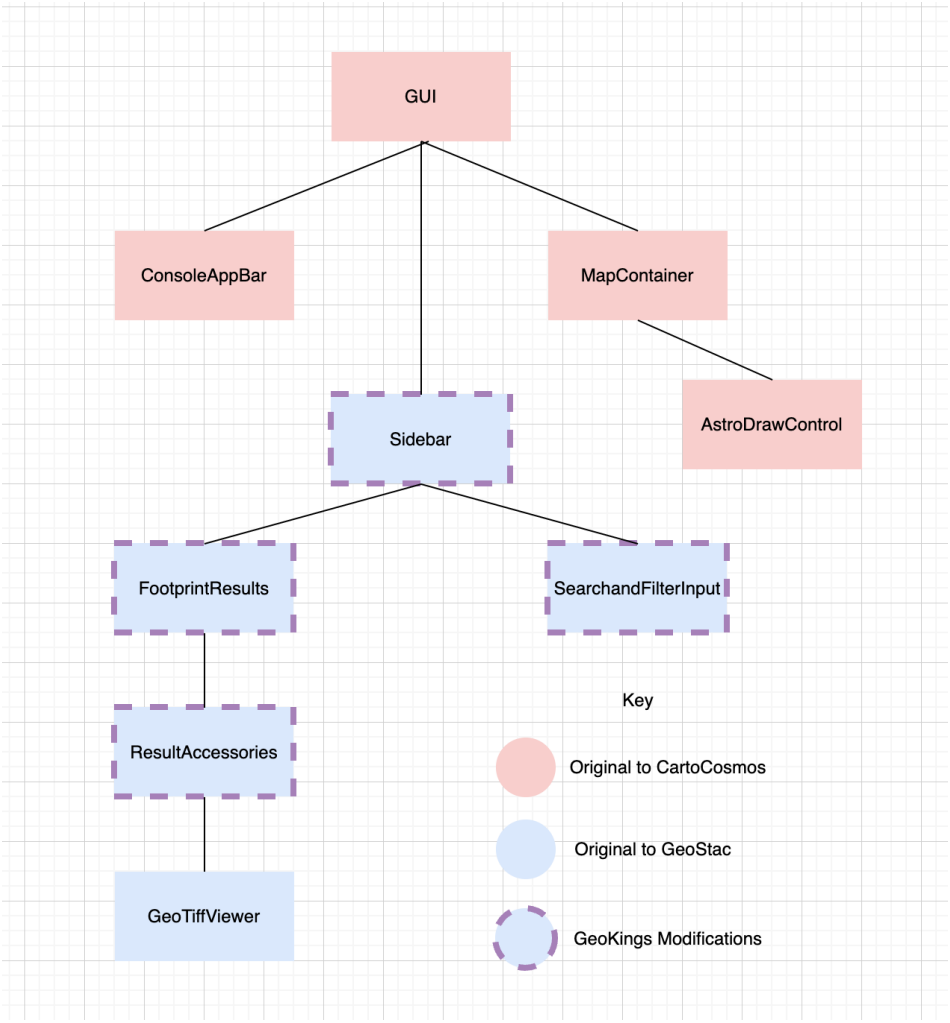

Figure 4: Diagram of GUI

a. Sidebar

The Sidebar is a component that is the parent component of Result Accessories and Search and Filter Input. Sidebar sets the visual parameters for the right part of the GeoSTAC application. Additionally, it is the component that calls both of its child components and holds each of them responsible for their functionality.

b. FootprintResults

The Footprint Results component is a child component to Sidebar. Its function is to retrieve all of the API links containing data in JSON format, organize it so that it can load the footprints properly, and then call Result Accessories with the organized API links. The modifications made to this component were the additions of the vector data API links. There are multiple checks performed inside the component to accommodate the different format of the vector data API.

c. ResultAccessories

The Result Accessories component is a child component to Footprint Results. Its function is to display each raster or vector footprint card on the right hand side of the GeoSTAC application. The modifications made to this component were focused on being able to format vector data footprint cards in a different format than how the raster data cards are displayed and filtered.

d. SearchandFilterInput

The Search and Filter Input component is a child component to Sidebar. The SearchandFilterInput's function is to allow the user to narrow down their raster/vector data results according to the parameter they would like to filter by. For raster data, a user can filter a footprint based off of its id, date, and location. The modifications made to this component was creating a filtering system through the use of checkboxes that filters all vector data based off of its "queryables". These queryables are all the core information given from the PyGeo API of the specific footprint selected.

e. GeoTiffViewer

The GeoTiff Viewer component is a child component of Result Accessories. It is a simple overlay that gives a better view of a footprint and it provides external links to give the user more information regarding a specific footprint. It is called through an HTML element in Result Accessories.

F. AstroDrawControl

AstroDrawControl is a child component of the map container. This component is what adds drawing controls to the Leaflet map. These drawing tools are what is used in order to select several footprints at a time on the Leaflet layer.

Implementation Plan

Through collaboration, many of the previously outlined modules are already drafted and are nearly ready for the final version. There are only a few components that still actively need development in order to be implemented with the bigger picture. Namely, those components are the Symbolize_with_icons module and the Remove_symbol module. These two modules have a handful of blockers that the team is currently working through in order to move forward with the development of the project. These modules are attainable for the team given a structured plan. Below is the

working timeline for the rest of the semester.

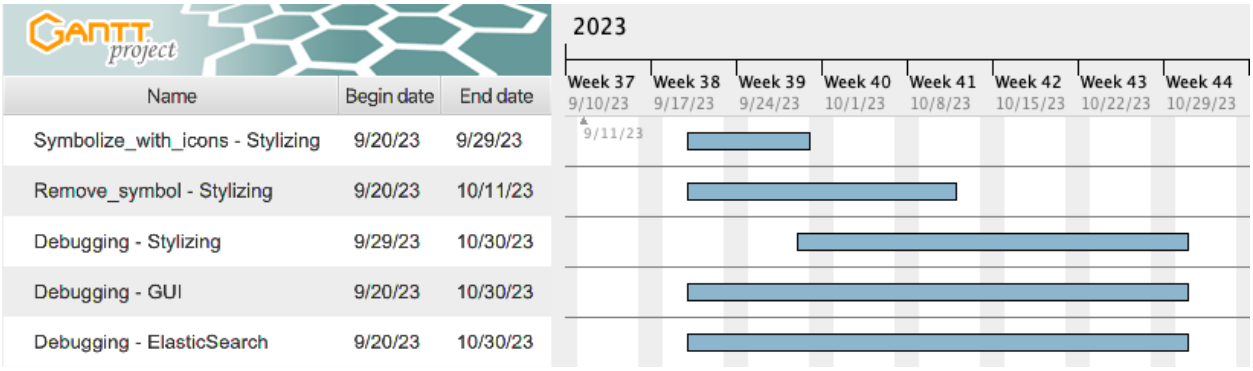

Figure 5: Gantt Chart of Implementation Plan

If all goes to plan, the two remaining modules will be implemented either by the 29th of September, or the 11th of October. The day the first remaining module is completed, the team will begin to wrap up the stylizing segment by debugging and polishing. This will allow the team to focus their efforts on getting the module operational before making sure that it is production ready. Outside of the topic of stylizing, the remaining segments include the debugging of the GUI and ElasticSearch. Because most of the programming was completed over the summer, the team is going to be focusing their efforts on debugging and polishing those segments in order to achieve a high quality product.

Conclusion

GeoSTAC's mission boils down to providing Analysis Ready Data to researchers around the globe. In order to do this most effectively, we must adapt to the ever growing amount of unprocessed data in regards to planetary systems. By improving upon the application already in place, we not only contribute to the innovation of the research being done today, but also to the research of others for years to come. The features that

will be implemented by the GeoKings will open the door to new possibilities for planetary research in a way that has yet to be done at this caliber. In order to do this, we must develop several modules to facilitate the availability of ARD. Broken up into three sections, there is the Stylizing, the API Fetch, and the GUI segment. Within Stylizing, Symbolize_with_icons and Remove_symbol must be developed. API Fetch uses FetchData and FootprintFetcher. For GUI, there is Sidebar, FootprintResults, ResultAccessories, SearchandFilterInput, GeoTiffViewer, and AstroDrawControl. Together, these modules will contribute to the advancement of planetary research. By having an outline for where we are and what point we need to be in the development of this project, we are able to effectively set milestones that allow for frictionless progress.