Technological Feasibility Report

GeoKings - GeoSTAC

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

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

CS 476

Dr. Leverington

4/07/2023

Northern Arizona University

Table of Contents

Topic	Page
Introduction …….	3
Technological Challenges …….	5
Technological Analysis ……..	7
I. Stylizing Vectors…….	7
II. Rendering Vectors …….	15
III. Search Feature ……	21
Technology Integration ……	27
Conclusion ……	29
References …….	30

Introduction

In today's era, digital data discovery has evolved to be completely online, with users discovering and sometimes analyzing data through various search engines, web maps, and FTP sites. This includes spatially enabled data that can be located within the context of a map. Some examples of this are restaurant locations, spatially enabled tweets, and satellite collected images.

The Astrogeology group of the USGS (United States Geological Survey) stands at the forefront of planetary data preservation, analysis-ready data delivery, and planetary data access. One of their focal points is to support community data exploration and availability. As part of this vision, the USGS has launched a project called "GeoSTAC," a web-based application that has been created and updated by preceding NAU capstone groups. GeoSTAC allows the serving of planetary data using community-developed standards to access Analysis Ready Data (ARD). ARD refers to data that has been processed through highly sophisticated imaging software, which significantly reduces the extent of data processing and obviates the necessity for the community to download large volumes of data for analysis and research purposes. The primary objective and use of the GeoSTAC application are to implement their focus areas outlined above by providing ARD to the global planetary science community, simplifying their research and rendering it more efficient. However, the current version of GeoSTAC supports only raster data APIs (a grid of values on a map), and not vector data (specific features on a map), which encodes data into a smaller file and is much better in regards to scalability, editing, and precision. The current use of raster data

API's results in a limited resolution for the GeoSTAC map and makes data challenging to modify.

Therefore, the USGS Astrogeology group is seeking a solution from the GeoKings team to (1) enable their vector data APIs within the GeoSTAC web application and re-release it with rich support for vector data visualization, discovery, and download. (2) To enhance the web application with the capability to perform advanced search operations using the Features API CQL (Common Query Language) specification, created by Open Geospatial Consortium (OGC) which is specifically designed for querying spatial data. This specification enables the integration of geospatial search capabilities with the ElasticSearch backend, allowing users to efficiently search and filter data based on location-based criteria. (3) To customize the visual representation of the newly created vector features, the application of Styled Layer Descriptor (SLD) files is proposed. These files are a set of rules that describe how to display different features on a map, making it easier to tell them apart from one another.

In this report, we will examine the feasibility of this project and how it could help the community in accessing and analyzing spatial data efficiently. In the next section we analyze the technological challenges of GeoSTAC by going over the high level requirements of our system. From there, we move on to the "Technology Analysis" section where we specifically discuss each of our major technological issues as well as design decisions. Next, we will transition to the "Technology Integration" section which

introduces our overall software architecture for our system and how each of the micro-solutions introduced will come together. Lastly, we outline and summarize everything that we have covered in this document.

Technological Challenges

The first challenge posed by this project is the rendering of the vector data on the map. The map is already integrated into the GeoSTAC web application, and it is our job to make the new vector data render on the map. Leaflet is a JavaScript library for interactive maps [1] that we are tasked with using. Learning how to effectively leverage Leaflet is a challenge that is necessary to overcome in order to ensure that the final product represents our abilities at the highest caliber.

Another challenge that we anticipate is search functionality within the GeoSTAC web application. Currently, there is a search function already implemented, but as of right now, it only supports the searching of raster data. The challenge presented to us is to implement updated searching functionality to support vector data as well. Familiarizing ourselves with this API will be the most challenging part, which is crucial to our success.

Stylizing the vector data will be another obstacle to overcome. Once the data is displayed, making sure that it is shown in an informative way will be the next step in our process. Stylizing the data will take a high level of comprehension on the topic of SLD

files. We will need to be sure that the files are appropriately handled for our project to succeed.

There will be the challenge of adding a tool to enable users to download specific features from the displayed web map. Another challenge will be to make sure that the work that we do is fully documented to the standards outlined by the Open Geospatial Consortium (OGC). Lastly, updating the user interface of the GeoSTAC web application to accommodate the additions that we made over the course of our project will be necessary.

- Key features will include:
    - The ability to render OGC Features API compliant vector data sources in the GeoSTAC webmap.
    - The ability to search using the OGC Features API Common Query Language (CQL) specification to the provided ElasticSearch backend.
    - The ability to stylize the vector features using Styled Layer Descriptor (SLD) files.
- Stretch goals will include:
    - The ability to select and download features from the GeoSTAC web map.
    - Developer focused documentation describing how other developers can make use of the library and APIs in their web map applications.
    - Updates to the UI as appropriate to support the visualization and discovery of vector data sets.

Technological Analysis

As a team, we have three ultimate tech challenges to research and deploy. These are Stylizing Vectors, Rendering Vectors, and using ElasticSearch. In order to understand how these technologies will fit together we did some research in order to find appropriate plugins or tools in order to keep everything compatible and up to USGS standard. In this section we will take the time to discover libraries or plugins that can be used together and are compatible. We will grade the tools and decide on a few to use. Here is what we found.

I. Stylizing Vectors

Since vectors are now able to be rendered on the map we need a way to be able to visually distinguish vectors from one another. A user should be able to look at a vector on a map and gain information about it through visuals alone. In order to give the user this feature we'll be implementing vector stylizing. Vector stylizing is the process of rendering individual vectors with symbols corresponding to the type of feature the vector is. Such as a canyon vector containing a symbol made of slashes, while lakes are displayed using a dot symbol.

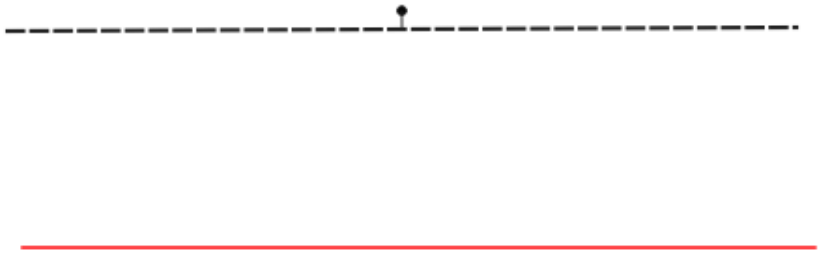

Figure 1.1 Styled black line (top) vs unstyled red line (bottom)

Through stylizing vectors we're able to gain advantages over plain rendering which include:

- Being able to use symbols to visually describe vector areas instead of only colors. The symbols would be assigned to certain geologic features, such as dotted lines being assigned to canyons, much like a physical map would have symbols and a legend to help its users to further understand what the area is like in terms of geography.
- Allows the user to quickly understand data associated with the vector area. Users won't have to do an extensive lookup on certain vectors to understand the vector they are interested in. They would just have to reference a legend to understand the symbol.

Displaying the stylized vectors is a two part problem which includes using an appropriate set of symbols and then displaying these symbols. We'll need two separate solutions to solve these problems which are:

- A package that contains symbols to display on the Leaflet map. This requires finding or creating a set of symbols in order to show them on a Leaflet map.
- A program that can then process GeoJSON data through a SLD file to produce a stylized vector. GeoJSON is a data structure for storing data on different types of geological features [2].

Since stylizing the vectors is a two part problem that requires two different technologies for a solution, we'll cover the first problem which is finding a suitable symbol set and its possible solutions. Following this we'll explore the second problem which is finding a way to display these symbol sets and its possible solutions.

For choosing a package of symbols to use with our program we'll examine a certain set of characteristics in order to determine if the package is compatible with GeoSTAC. These characteristics are as follows:

- Set-up Time: Saving time is important to ensure that the team stays on track. In order to keep on track the team should make sure that the tools we are using don't require more time than they can afford to spend.

- Visual Clarity: Since this map will be used by many to look for and discover data, it's important that the symbols are easily visible. This way users are able to distinguish certain vector features from one another.

- Small Size: The size of these files should be deemed to be small enough to quickly display to the application. Too big of a file size could cause more loading time ultimately slowing down the GeoSTAC application.

For displaying these symbol sets to the Leaflet map we have two options we'll introduce:

- Our first option is called Geologic Symbols, which is a framework that contains geologic symbols with a SLD to use with various mapping programs such as Leaflet.
- Our second option would be to use FGDC Geologic Patterns for the Web, which is a framework mainly composed of symbols that we can use with a custom made SLD to display these symbols.

The Geologic Symbols package was recommended to us by our sponsors. It's composed of a small team and is hosted on GitHub for anyone to freely. This package provides many different geological symbols in many different file formats including SLD files. Since the developers have provided many different file types to increase compatibility with many mapping services, it can be expected that it has been used in mapping projects. Because the package comes with an SLD file type it can even be used with Leaflet with some special tweaking.

Another option for using geologic symbols would be to use FGDC Geologic Patterns for the Web. This was included in the previous GitHub of Geologic Symbols,

explaining how it was a similar style of project. There have been a few updates to the program since its release so it can be assumed the developers are actively supporting the package. The package contains mainly PNG's and SVG's (Scalable Vector Graphics) of different styles of geologic drawings. SVG's are images that are able to be scaled up or down at any resolution without losing quality, unlike other image formats such as PNG and JPEG [3]. Examples of projects using the tool aren't readily available so we will have to experiment to get it to work.

To fully understand what package would be best for our project, we'll begin to analyze each package based on the desired characteristics. Once again the desired characteristics are set up time, visual clarity and small size. We'll begin by first analyzing Geologic Symbols and how they meet these desired characteristics:

- In terms of being able to use the tool quickly, this package does very well. The package already has symbols set up with an SLD file. This means we will not have to spend time creating our own SLD file to use with the symbol set.
- For visual clarity many of the icons are easily visible and the image quality doesn't appear to be compromised. The SVG images are mainly black and white which would help with contrast on most of our Leaflet maps. If a map has a lot of black and white, it could be an issue since these symbols could get lost in the background.

- In terms of image file size this package has files that are on average around 2KB - 4KB large. These file sizes are very small and even with multiple of them appearing at once shouldn't slow down the speed at which the vectors are displayed.

Now we'll take a look at how using FGDC Geologic Patterns for the Web compares.

- This package doesn't come with an SLD file, so the team would have to make one. Making an SLD file to use with these symbols will take more time.
- For visual clarity this package is about the same Geologic Symbols. It mainly contains black and white images with some red and blue symbols. The image quality itself seems to be clear and not blurry. And the issues of possible color overlap with the map is an issue with this package as well.
- The average size of a symbol for this package is around 20KB, which is still relatively small.

When it comes to choosing a symbol set to use, with Geologic Symbols, we'll be able to spend less time on setting it up. It also has good visual clarity, and contains very small files. With FGDC Geologic Patterns for the Web we'll have to spend more time setting it up, but it won't be a huge impairment, the visual clarity is also good, and the size of the files are small.

After doing some deliberating the team has decided to go with the Geologic Symbols package. In reference to Table 1.1 Geologic-Symbols either ties or beats FGDC Geologic Patterns for the Web in every category making Geologic-Symbols the clear winner. The biggest differences being Geologic Symbols requires less time to get working and the size of the average file is five times smaller than files in FGDC Geologic Patterns for the Web.

	FGDC Geologic Patterns for the Web	**Geologic-Symbols**
Time setup	2	5
Visual clarity	5	5
Size	3	5
Total	**10**	**15**

Table 1.1

Now, for displaying these symbols, we need another tool. Due to the nicheness of the challenge we have, which is processing GeoJSON data with SLD files to display symbols to a Leaflet map, the available tools are extremely limited. With that said, we still require the limited tools to be able to meet certain characteristics in order to use it in our project. These desired characteristics for displaying these symbols are as follows:

- Speed: Using a slow website not only frustrates the user but is a detriment to their time. For this reason we find it important to use a tool that will not slow the website's processing time, in order to give the users the best experience possible.

- Compatibility: In order for a tool to properly display stylized vectors it will need to be compatible with GeoJSON objects and SLD files.

Once again due to the nicheness of our problem there is only one tool readily available the team has found called “Leaflet.SLD”, which was recommended to us by our sponsors. The program itself contains a very small codebase and has only one contributor. Leaflet.SLD is compatible with the current version of Leaflet, which means the team wouldn't have to worry about fixing any compatibility issues. Once again finding example projects using Leaflet.SLD is not readily available. Since the program is a speciality program designed to fix one specific problem, it is not expected to have seen a lot of use.

Even though there is only one option we’ll begin to analyze Leaflet.SLD to make sure it suits our project. If the package doesn’t meet the desired characteristics the team will have to look into making a custom program. We’ll take a look at Leaflet.SLD and how it fairs in speed and compatibility.

- Leaflet.SLD contains a very small codebase consisting of about 300 lines of code. It is also written in Javascript which is a relatively fast language.
- SLD files and GeoJSON objects work with Leaflet.SLD by default. Taking in the GeoJSON object and SLD file, then rendering the styled vector to the Leaflet map.

While Leaflet.SLD is the only readily available option we've still found it to be a highly desirable tool. After the team had looked over the codebase they deemed the program to be acceptable and didn't contain any signs of slow down. The codebase is also relatively small meaning the team could easily make any changes to the program in order to optimize speed. Leaflet.SLD works with GeoJSON and SLD files by default, meaning that it is perfectly compatible with the GeoSTAC project as is.

	Leaflet.SLD
Speed	5/5
Compatibility	5/5
Total	10/10

Table 1.2

Even though Leaflet.SLD is the only available option right now the team has found it to be an ideal tool for the project. The program scored flawlessly in compatibility and speed, proving again to be an ideal tool for the team's project, see Table 1.2. The process of creating a new program to use with the project is deemed unnecessary and fruitless after this analysis of Leaflet.SLD.

II. Rendering Vectors

Vector data, a type of geological mapping data, is stored as a collection of polygons and shapes associated with specific geological features. This is significant because the current iteration of our web application can only display raster data, which

is a rendered image that uses an array of pixels to create a visual representation of the map. However, raster data does not provide insight into specific features such as mountain ranges, volcanoes, mineral deposits, and others. Our goal is to update the web application to support vector data, allowing users to research geological features on a range of planetary bodies more effectively.

Rendering the data for the web application will be done using an API that retrieves vector data in the form of a GeoJSON file. This file will contain the coordinates and data for these features. To visualize this data in the form of symbols, shapes, and lines, we will use Leaflet, a basic mapping library for JavaScript. In terms of compatibility, we have identified two main options for rendering vector data: Mapbox GL, a vector tiling library, and Leaflet.VectorGrid, a plugin for Leaflet that handles vector tiling and styling(5,6). Before selecting the best-suited solution, we have established the following criteria:

- Compatibility: The solution must be compatible with our current technologies, such as Leaflet and JavaScript, to ensure the web application remains cohesive and performs as expected.
- GeoJSON Support: The solution must be able to use GeoJSON and SVG files, which is crucial for proper rendering of vectors and the overall functionality of the web application.

- Ease of Maintenance: The solution should be easy to maintain and update, which is important for the longevity and sustainability of the web application once it is handed over to USGS.
- Environment and community: We want a solution with an active community for better documentation and support if we encounter any issues.

Now, let's compare Leaflet.VectorGrid and Mapbox GL, both of which are compatible with Leaflet and JavaScript. These libraries cater to different needs and have varying levels of complexity, but both are capable of using GeoJSON and SVG files, which is essential for our project. Developed by Iván Sánchez Ortega, Leaflet.VectorGrid is a lightweight plugin with a mature codebase, while Mapbox GL, developed by Mapbox, is a more advanced library(5,6).

Leaflet.VectorGrid

- A lightweight and easy-to-use plugin for displaying vector tiles in Leaflet-based maps
- Developed by Iván Sánchez Ortega
- Well-established with a relatively mature codebase
- Already being used in our web application
- Easy SVG integration
- Average documentation

Mapbox GL

- A more complex library that provides advanced features for creating interactive maps
- Developed by Mapbox
- First stable release in 2016 but gained popularity due to its versatility and ease of use
- Contains a plugin called Mapbox.Leaflet that makes both the rendering and styling using Leaflet compatible
- Difficult SVG integration because it needs to be converted into a data URL in the form of a JSON
- Good documentation

We have to choose one based on several pros and cons. Here are our conclusions:

- Compatibility and Cohesiveness: Both Leaflet.VectorGrid and Mapbox GL are compatible with Leaflet and JavaScript, which are the existing technologies used in our project. However, Mapbox GL requires a separate plugin called Mapbox.Leaflet and SVG files to be converted into a data URL(data sent over the URL instead of an external file), which could reduce cohesiveness and future compatibility. Leaflet.VectorGrid, on the other hand, is already being used in the application and has full support for our existing libraries without requiring modifications, making it a more cohesive choice.

- Ability to use GeoJSON files: Both options can handle GeoJSON files, which is an essential requirement for our project.

- Licensing cost: Leaflet.VectorGrid is an open-source plugin with no licensing cost, while Mapbox GL is a commercial product that may require a paid license for certain uses. At this time, we are unsure if a cost will be required since the code will be open-source. Mapbox's licensing may be free in a free-to-use application.

- Performance: Mapbox GL is known for better performance when rendering large datasets, while Leaflet.VectorGrid has some limitations in handling very large datasets. However, for the scale of our project, either option should suffice.

- User base: Both options have a significant user base, with Mapbox GL being used by companies like Strava and Airbnb, and Leaflet.VectorGrid being used by organizations like NASA and the USGS. An advantage of using VectorGrid is that USGS has already implemented it into our current system.

- Maturity of technology: Mapbox GL is a mature and established technology, with a long history of development and continuous updates. Leaflet.VectorGrid is a relatively newer technology but has been actively developed and maintained.

	Leaflet.VectorGrid	**Mapbox GL**
Compatibility	5	3
Licencing	5	3
Performance	3	5
cost(time)	5	3
User Base	5	4
Total	23	18

Table 2.1

Based on our analysis, we can conclude that both options are suitable for our project, but Leaflet.VectorGrid has a slight advantage in terms of compatibility and cost, while Mapbox GL has an advantage in terms of performance and maturity. Ultimately, we have chosen Leaflet.VectorGrid since it scored much higher in Table 2.1 and has already been incorporated into our application and has absolute compatibility with other Leaflet plugins currently in use. In addition, the cost of making adjustments to make Mapbox viable is just far too much.

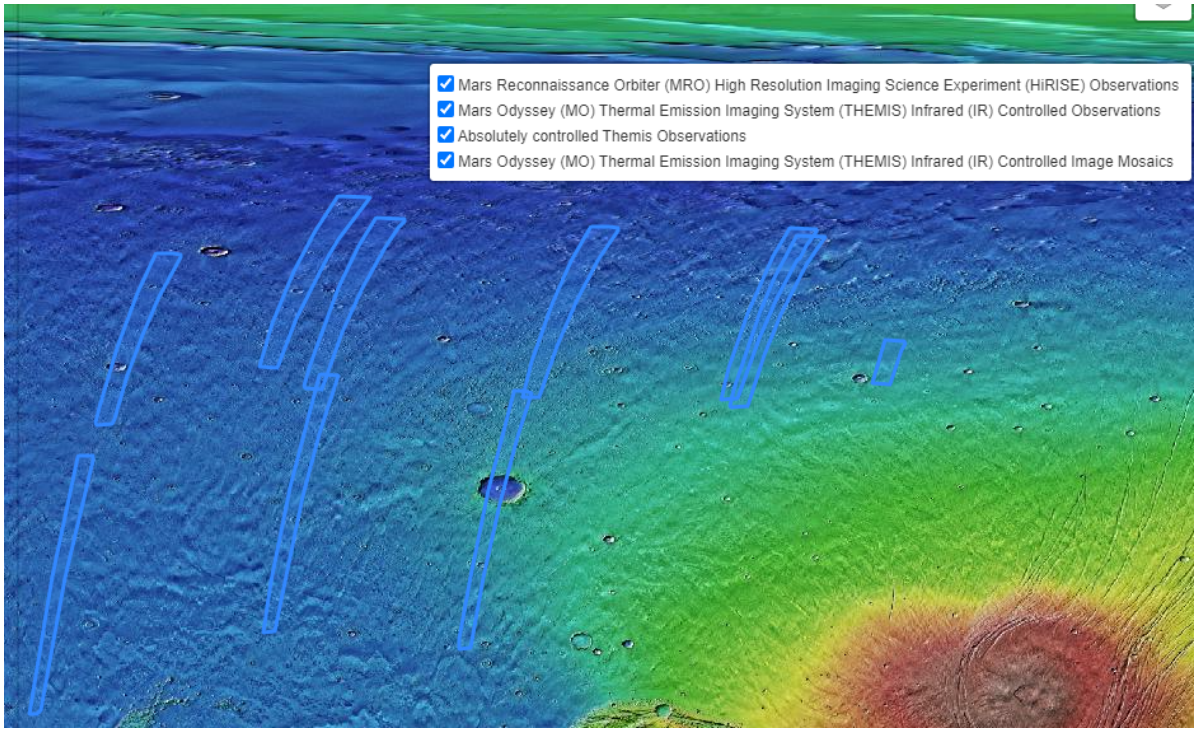

Figure 2.1

As shown in Table 2.1, we have chosen to continue using Leaflet.VectorGrid over Mapbox. Looking at Figure 2.1, here is an example of what the polygons on the planetary bodies will look like once we render the vectors. Figure 2.1 is taken directly from the GeoSTAC website, as Mars has dummy values to demonstrate the feasibility of displaying these vectors. Our job will be to continue using this in combination with a given API to collect GeoJSON files that we can then render onto the map.

III. Search Feature

The inclusion of a database of vector data, and the ability to display this information, necessitates a method of finding the desired data. For this additional data to be useful, a user needs to be able to select the vector data that is meaningful to them and to then be able to query the database to find data that matches the selected criteria.

Without this search feature, the only other approach would be to display all the vector data for a given map by default. This approach would have two primary problems:

- The first is that querying the database for all available vector data of a given map and then visualizing all of that data would unnecessarily waste both bandwidth and CPU processing time. As a result this would be an inefficient system - albeit one that would provide the desired vector data.

- While this approach would successfully display the desired data, it would also display all the undesired data as well. The maps produced by this method would be cluttered and confusing for the users and potentially impede their ability to find the data relevant to their research.

With this in mind, our approach will be to provide functionality to the user that will allow them to search the database for specific vector data - this will be done in conjunction with the necessary UI improvements such as check boxes - to indicate the specific geologic features that are being looked for (mountains, volcanoes, rivers, etc…).

There are several features that will need to be possessed by any possible solutions if they are to provide the necessary search functionality. These are listed individually in the following paragraphs, along with a brief description:

- Speed: A slow response time when using a web application can be a very frustrating experience, and one that is important to avoid. Given that the GeoSTAC web application works with maps and imaging, this is doubly true as those files can already be large and slow to load. Our ideal solution will be to maintain the current server responsiveness while searching the database for this additional layer of data.

- Database Compatibility: This project will be working with an existing database of raster and vector data, accessed through an API. This database retains information in the GeoJSON format and any search functionality will need to be compatible with this format.

- Standards Compliance: In addition to working with an existing database of map information in the GeoJSON format, this search feature will need to be able to accept queries submitted according to the CQL standard that this project must adhere to. This standard is a requirement from the sponsor.

- Redundancy: Lastly, as a web application that provides access to scientific data that is necessary for researchers - it is important that access to this data is reliable. As a result, it is important to account for any potential outages or server failures that could result in a partial - or complete - inability to search the database. Any search function operating on the backend of the web application

will need to provide some level of fault tolerance and redundancy so that in the event of an outage, application users will still be able to access the full data set.

While other search engines that could be built into the backend of an application do exist, there are no viable alternatives to the use of ElasticSearch. In this case - as this is an existing web application - a functioning ElasticSearch backend is already in place, and this means that replacing it unnecessarily would require our team allocate a great deal of resources to this task. The sponsors of this project have also stated that they wish for ElasticSearch to continue to be used as the search engine for the GeoSTAC application. This being said, it is worth discussing in the following section what ElasticSearch is - both generally and then more specifically - and why it is the correct solution for this project.

In the broadest sense, ElasticSearch is a search engine and performs the same job as the engines we are all familiar with; Google, Bing, DuckDuckGo, etc.. This search engine is meant to sit on the backend of the web application, accessed by users through an application, see figure 3.1.

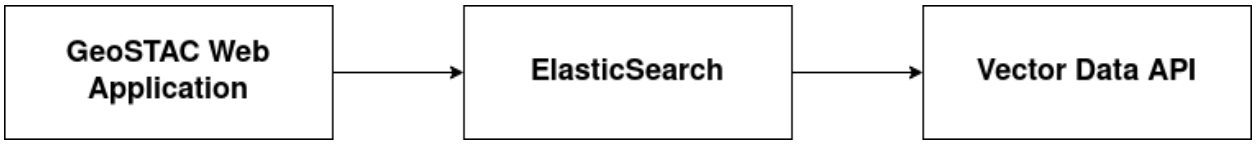

Figure 3.1

ElasticSearch is designed to handle a wide range of data - including geo-spatial data - which is what is of interest for this project. Developed in 2010, ElasticSearch has been available for over a decade, allowing for wide adoption and community support to

develop. The foundation of ElasticSearch is the Lucene search engine, developed and maintained by the Apache Foundation [7]. Built on top of the Lucene engine are a number of additional features designed to improve scalability, responsiveness, fault tolerance, and redundancy.

- ElasticSearch utilizes a “word-level inverted index”. An “inverted index” is the most widely used structure for a full-text search engine such as Lucene, and is generally considered to be one of the fastest ways in which to retrieve data [8].

  An “inverted index” is a data structure in which any stored document is scanned and the exact location of this document - or elements of that document - are stored in an index file.

  In the case of a “word-level” inverted index, each uploaded document is scanned and each individual word of the document is stored in the index, along with all of the locations of that word within the database. This allows for rapid searching and retrieval of the database.

- A solution requirement is for easy compatibility with the current database and for easy API access. ElasticSearch has a number of different APIs available that allow for a wide variety of queries in order to monitor not only the performance of the engine itself but to also access the data it has indexed.

  ElasticSearch has also implemented structures specifically to handle geo-spatial data [9]. The GeoShapes data structure, for example, allows the engine to index not only rasterized map data - meaning simple, individual points

on the map - but also shapes like the ones created by the vector data being added to the GeoSTAC application.

- Two features built into ElasticSearch are “nodes” and “shards” [10]. Nodes are effectively the individual servers on which ElasticSearch is running. Within each node there will be one or more “shards”, which are individual instances of ElasticSearch running on those servers or “nodes”. Running multiple shards - or instances of ElasticSearch - allows for improved server responsiveness since multiple simultaneous requests to the search engine can be split between the shards allowing them to work in parallel. To provide redundancy, an update-to-date copy of each shard - known as a “replica” shard - is stored on a different node from the “primary” shard. In the event that a node goes offline, all data can still be accessed as the replica shard will take the place of the primary shard that is now unavailable. This structure allows for ElasticSearch to have scalability by adding additional servers if needed, and to be fault tolerant by providing redundancy for each shard should one of those servers fail.

In our case, the approach is predetermined - we will be querying the vector data using an ElasticSearch backend as that is a request of the sponsors, and the backend is already in place and functioning with the raster data currently available on the GeoSTAC application. The analysis of ElasticSearch by our team however supports the request by our sponsors to utilize it. As has been shown in the prior sections, an ElasticSearch backend is capable of providing all the necessary qualities of speed,

compatibility, compliance, and fault tolerance. Additionally, the majority of the frame-work for querying the database through ElasticSearch is already in place, our team expects that we will only need to augment the current application design to handle queries of the more complicated vector data.

In order to validate our expectations that ElasticSearch will be up to the task of handling both raster and vector data, our team plans to test this by writing simple queries - using the OGC Features - API CQL specification - and to see if results are returned.

Technology Integration

Now that all of the technological challenges have been addressed, we need to address how to integrate all of our chosen solutions together, see figure 4.1 for a visual representation of how the different technologies will be workin with each other. Firstly, it’s important to note all of the technological solutions depend on the vector API. Also of all of the technological challenges, only vector rendering and vector styling will be working directly with each other.

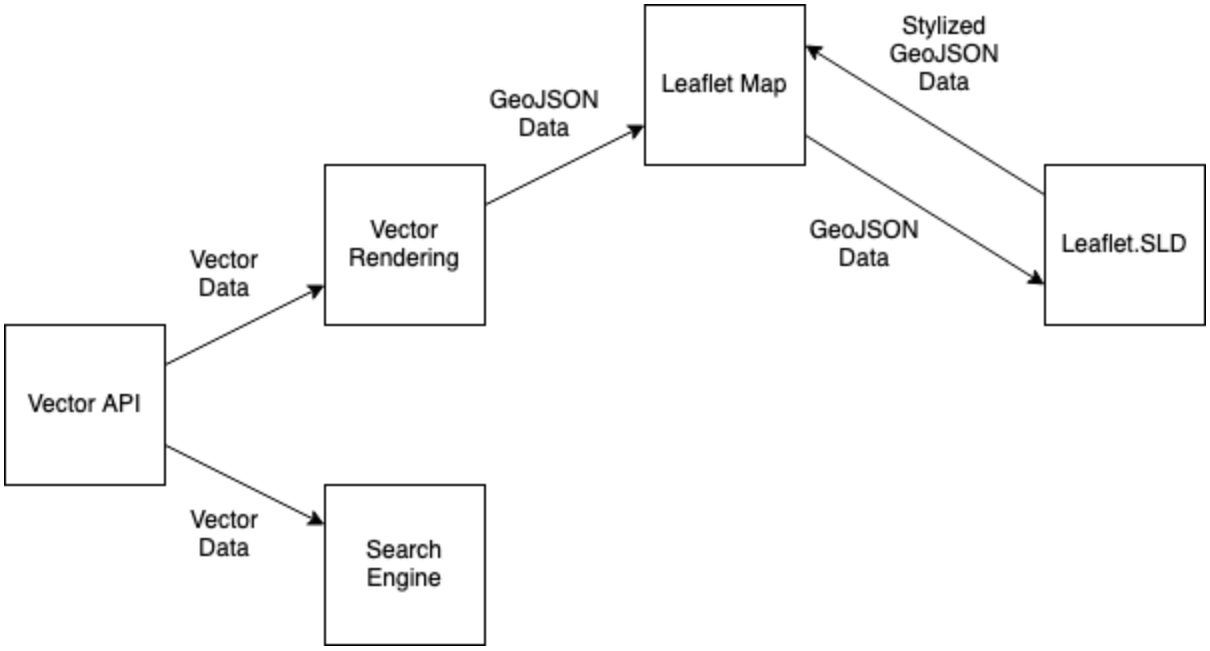

Figure 4.1

The vectors will be processed using data from the vector API, processed into GeoJSON data and then rendered to the Leaflet map. From here the GeoJSON data will begin to be stylized by Leaflet.SLD scanning for any new vectors, referring to that GeoJSON's data, and then visually styling the vectors according to their associated features.

The search feature will integrate with this process solely through the vector API. A GeoSTAC user will select the features they are interested in (mountains, lakes, rivers, etc…) and then pass that information to the ElasticSearch backend. ElasticSearch will then query its Index to locate the desired data. After the data has been found, ElasticSearch will retrieve the requested vector data and pass it into the vector API for rendering and eventual styling.

Conclusion

The features developed by GeoKings will ultimately allow for a more accurate, user-friendly experience for those looking to analyze spatial data. Using the tools provided by the JavaScript library Leaflet, we are able to display vector data to the GeoSTAC web map in addition to the raster data as well as stylize the data in a way that is understandable to the users. By adding updated searching functionality to ElasticSearch to support the implementation of visualized vector data, having access to spatial data will now become even more accessible. Moving forward, it is our goal to deliver on these topics through diligent communication and effort, providing our client with the best product we have to offer.

References

1. An open-source JavaScript library for interactive maps. Leaflet. (n.d.). Retrieved April 4, 2023, from https://leafletjs.com/
2. Geojson. GeoJSON. (n.d.). Retrieved April 4, 2023, from https://geojson.org/
3. SVG: Scalable Vector Graphics. MDN. (n.d.). Retrieved April 4, 2023, from https://developer.mozilla.org/en-US/docs/Web/SVG
4. *Styled layer descriptor*. Open Geospatial Consortium. (2023, February 21). Retrieved April 4, 2023, from https://www.ogc.org/standard/sld/
5. *API reference: Mapbox GL JS*. Mapbox. (n.d.). Retrieved March 23, 2023, from https://docs.mapbox.com/mapbox-gl-js/api/
6. Leaflet. (n.d.). *Leaflet/Leaflet.VectorGrid: Display gridded vector data (sliced geojson or protobuf vector tiles) in leaflet 1.0.0*. GitHub. Retrieved March 23, 2023, from https://github.com/Leaflet/Leaflet.VectorGrid
7. *What is ElasticSearch?* Elastic. (n.d.). Retrieved March 22, 2023, from https://www.elastic.co/what-is/elasticsearch
8. GeeksforGeeks. (2023, March 13). *Inverted index*. GeeksforGeeks. Retrieved March 22, 2023, from https://www.geeksforgeeks.org/inverted-index/
9. *GEOSHAPE field type: ElasticSearch Guide [8.6]*. Elastic. (n.d.). Retrieved March 22, 2023, from https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-shape.html
10. George Bridgeman. (2021). *What are ElasticSearch shards? Why do they matter? ElasticSearch cluster architecture explained.* Retrieved March 22, 2023, from https://www.youtube.com/watch?v=NxpZyQVO0K4.