Contents

1 Introduction

An integral facet of space exploration is creating and launching satellites into space to explore other planets and collect data. This data is crucial to increase our understanding of the solar system and universe as a whole. Some of this data are images taken of a planet's surface and are the foundation with which to generate maps of the planet's surface. Today, all eyes are set on Mars which has been chosen as the next stop for manned space exploration. To accomplish this goal, one of the initial missions to Mars was the 2001 Mars Odyssey, commissioned to explore the red planet and transmit its findings back to Earth. It has orbited Mars since October 2001 and is still actively providing essential data today. The Odyssey's primary goals are to provide spatial data of the planet's surface, image surface minerals, and locate potential signs that the planet may have once supported life. Without this orbiter, and the system of maps it has gifted to us, colonization would be impossible.

Overall, there are many tools scientists use to create these maps; however, there are not many well-developed tools to view them virtually and account for the various viewing options. The United States Geological Survey (USGS), the CartoCosmos' client, is contracted by the National Aeronautics and Space Administration (NASA) to support the planetary science community. NASA has assigned USGS the task to create a new tool to view planetary maps with an emphasis on usability and accuracy.

1.1 The Problem

Trent Hare, a research developer and data processor, and Scott Akins, an IT specialist, are two employees from USGS who have assigned the team the task of overseeing upgrades to USGS's current virtual planetary mapping viewer. The USGS team is currently using OpenLayers v2.0, an open-source mapping tool, for their implementation. It has been providing map visualizations to researchers for some time, but it has some flaws that need to be addressed. The four main drawbacks are:

- OpenLayers does not support planetary mapping.
- All of the planets and bodies have the same radius, Earth's, because its intended use is mapping Earth and is hard coded in the source code.
- The data is modified based on mathematical transformations and projections to convert the planet or body's various measurements to Earth's, causing the data to be inaccurately represented.
- The current code base for it is not modular, i.e., transformation functions are integrated into the OpenLayers source code and is not supported by other mapping applications.

The overarching goal of this project is to correct these issues and facilitate development of the first of many open-source mapping applications for USGS, NASA, and the planetary science community.

1.2 Our Solution

CartoCosmos will be creating a JavaScript node package containing the required mapping transformations to support different projections that can be used across multiple mapping applications. The team will be using Leaflet, an open-source mapping tool, to create a GUI containing the correctly-projected virtual maps. Users will be able to zoom, move the location of the view, select an area on the map, see gazetteer names and symbols on the map, use variable radii, convert between 0:360 and -180:180 longitude ranges, convert between planetographic and planetocentric latitudes, and load data layers. The application will connect to the USGS web catalog containing data on their maps and have an auto-complete search function that will help users locate different surface features by name. In the section below, we identify the key technical challenges we will face when implementing our solution for this project.

2 Technological Challenges

Many of the design decisions for this project have been made by USGS beforehand. These requirements are intended to solve the problems with OpenLayers and originate from feedback from the planetary science community. Some of these requirements include:

- Use the open-source mapping framework Leaflet to create the maps.
- Ensure compatibility with Jupyter Notebook, an easy-to-use interface for scientists to use the data.
- Use the JavaScript programming language.
- Follow Leaflet's plugin requirements to become an official plugin.
- Follow the Web Map Service (WMS) and Web Feature Service's (WFS) standards for querying data from the USGS web catalog containing planetary data.

This leaves us with four different design decisions we see as technical challenges:

1. GUI layout: how to create the GUI layout to ensure usability across a wide range of users with varying skills and knowledge
2. Projection algorithms: how to package the projection algorithms so that they can be used across multiple mapping applications.
3. Coding tools: what tools to employ to assist production, enforce code quality and style standards, and provide automation of trivial tasks
4. Search with auto-complete: how to implement a search feature to access Web Feature Service (WFS) information about the current planet or body being viewed by the application

These challenges represent distinct decisions that must be made to succeed in solving each one. To deliver the best possible final product, these decisions must be made objectively, with research and testing to back our choices. Below is our analysis of a number of possible solutions for each of these technical challenges.

3 Technology Analyses

For our technology analysis, we will be providing more information on each of the design decisions presented above. Each decision will have an introduction paragraph describing the problem alongside metrics we will be using to choose the best one. The metrics will vary based on the design decision being discussed. We will then describe the solutions we have to choose between, the pros and cons of each, and how they are rated under our metrics. We will also be giving examples of how we measured these metrics and tested the solution. After all potential solutions for a design decision have been inspected, we will summarize how each solution did by putting the ratings into a table. Finally, we will discuss which solution who chose and why.

3.1 GUI Layout

The Graphical User Interface (GUI) allows for users to connect and interact with data using visual indicators and graphical elements. An effective and easy to use GUI can make or break any product. Since the project is a virtual mapping application using Leaflet, the GUI layout must house a Leaflet interactive map. The GUI must also list the name of the planet and provide a mechanism to choose between different layers, projections, longitudes, latitudes, and scale options. In order to determine the best solution to this challenge, the team judged several different approaches based on distinct metric observations. Since the project is being designed for researchers and scientists, the GUI must be tested and validated by them for ease of use and familiarity. At this point in time, the team does not have any current implementations for the GUI layout so there is no way of objectively measuring these factors. The team will be flexible with the new GUI layout upon completion and issue updates in response to community feedback. The metrics the team used in order to decide the approach for implementing the GUI layout are:

- Time required to implement: Time in minutes
- Is it currently being implemented?: Yes/no
- Team familiarity: 0-4 team members

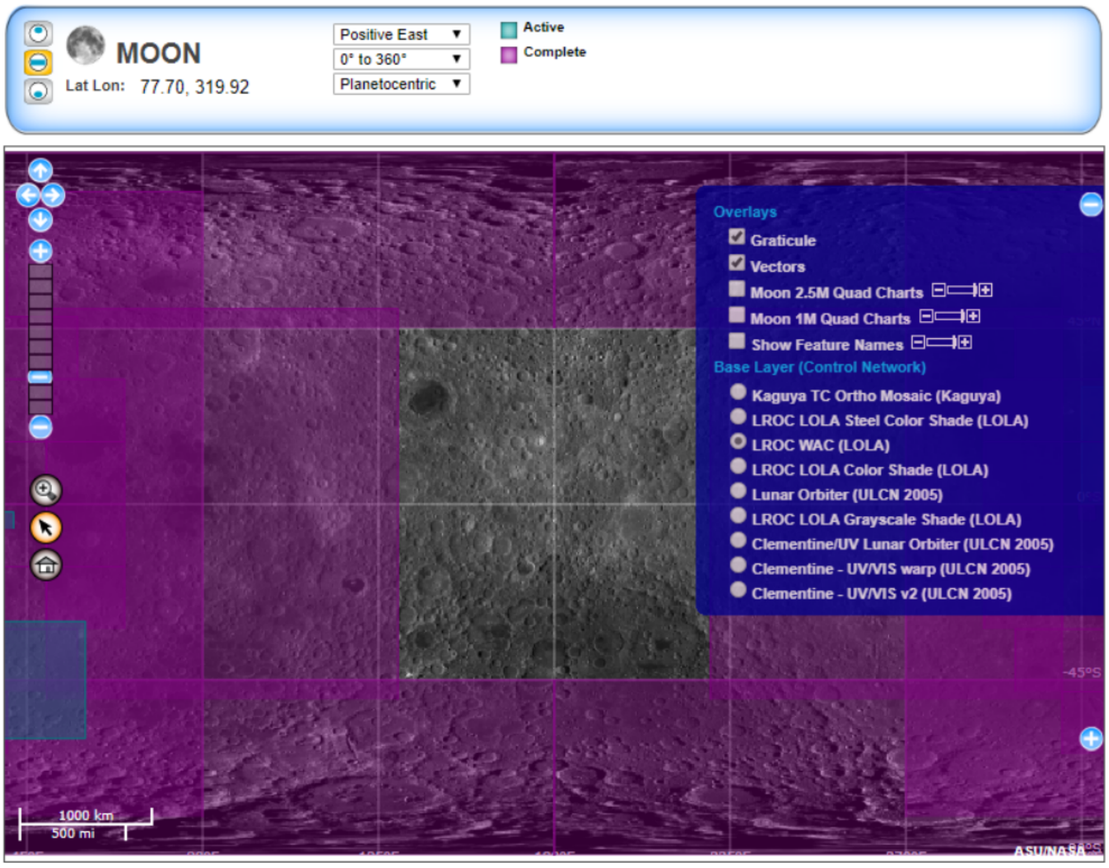
These metrics will allow us to judge the three different approaches for resolving the GUI layout with objectivity. The first metric will help us base whether the approach is going to take a long time to implement. We do not want an approach that we will not have the time to complete. The second metric defines whether or not the given approach is currently implemented by USGS, which signifies user familiarity. The last metric determines how comfortable the team feels with the the given approach. It was calculated by counting the number of team members who identified as being familiar with the approach.

3.1.1 Potential Solutions

The three potential approaches that the team evaluated to successfully implement the GUI layout are:

- USGS's current Implementation: using a combination of bounding boxes and OpenLayers' GUI tools to position the map and included features.
- USGS's Astro library with Leaflet GUI tools: using the Astro JavaScript library with provided bounding box functions and Leaflet GUI tools to create a new and customized, yet user-familiar, layout.
- Leaflet's GUI tools: using Leaflet GUI tools alone to develop a new layout for a completely new experience with complete control over design.

1. USGS Current GUI Layout



Lat Lon: 77.70, 319.92

Positive East ▼
0° to 360° ▼
Planetocentric ▼

✓ Graticule
✓ Vectors
Moon 2.5M Quad Charts
Moon 1M Quad Charts
Show Feature Names

Kaguya TC Ortho Mosaic (Kaguya)
LROC LOLA Steel Color Shade (LOLA)
[✓ LROC WAC (LOLA)]
LROC LOLA Color Shade (LOLA)
Lunar Orbiter (ULCN 2005)
LROC LOLA Grayscale Shade (LOLA)
Clementine/UV Lunar Orbiter (ULCN 2005)
Clementine - UV/VIS warp (ULCN 2005)
Clementine - UV/VIS v2 (ULCN 2005)

Figure 1: USGS OpenLayers Map Example

The first approach involves continuing the use of the GUI layout USGS has on their current implementation (Figure 1). USGS's current format uses a combination of different JavaScript coding libraries named Astro. Astro combines different HTML and JavaScript form entries to create bounding boxes which establish the GUI dimensions. The bounding box functions also create the projection buttons and the latitude and longitude drop-down menus, which are at the top of Figure 1. USGS used the OpenLayers GUI tools to create the scale bar, which zooms in and out, and the layer list. The layer list is opened by default on the right side of Figure 1 and can be closed by clicking on the minus sign. The one change to this GUI layout that we would need to implement is it would need to encase a Leaflet interactive map instead of an OpenLayers map.

In order to test how long it would take to implement this approach, the team used USGS's bounding box functions and the OpenLayers GUI tools to build a simple Earth Leaflet map. The implementation process was quite hard to understand because of the lack of examples showing how to use the OpenLayers GUI tools with a Leaflet map. This process took the majority of the implementation time because USGS already had connected the bounding box functions with the OpenLayers GUI tools. After getting a simple Earth map to connect the team found that it took 137 minutes to implement. Afterwards, only two of us felt comfortable working with Openlayers and USGS's bounding box functions. The following rankings were determined:

- Time required to implement: 137 minutes
- Is it currently being implemented: Yes
- Team familiarity: 2 team members

2. USGS's Astro with Leaflets GUI Tools

The second approach involves the use of the Astro JavaScript libraries, which are described above, combined with Leaflet's GUI tools. Using the bounding box functions will allow for reuse of the bounding boxes that contain the projection buttons and latitude and longitude drop-down menus. Using the Leaflet GUI tools, the team created a redesigned layer selector, scale bar, and zoom tool. Connecting the Leaflet GUI and the interactive map to the bounding boxes completed the GUI layout.

To test how long it took to implement this GUI layout, the team created and encapsulated a simple Leaflet map of Earth inside of the bounding boxes using Leaflets GUI tools. The team followed one of the numerous tutorials found on Leaflet's website. These tutorials help cut the total implementation time down significantly, with the majority of the time spent connecting the bounding boxes to Leaflet's map. This entire process took only 113 minutes to complete. Thanks to the plethora of Leaflet tutorials available online, three team members felt confident about this approach, while the remaining team member did not feel comfortable using USGS's Astro libraries. The ratings are:

- Time required to implement: 113 minutes
- Is it currently being implemented: No
- Team familiarity: 3 team members

3. Using Leaflet GUI Layout

The final potential solution involves designing a new GUI layout for the virtual maps by using only the Leaflet GUI tools. The Leaflet GUI tools possess many different interactive features that can be incorporated into the final product for added usability. The tools also provide user-friendly menus and buttons, which the current implementation of OpenLayers does not. Using only the Leaflet GUI was a far simpler approach since the team was able to abandon the Astro JavaScript libraries. This gave us more time to focus on other requirements and challenges.

To figure out how long this approach would take to implement, the team created a simple Leaflet map of Earth with an added-on menu. This was done by going through the tutorials on Leaflet's website. This was by far the quickest implementation because everything we needed to implement the GUI layout was in the tutorials. The process only took a total of 68 minutes. The one drawback of this approach is the need for customization of the GUI to add buttons and selectors for the viewing requirements. The team familiarity with Leaflet is high and earned the best team familiarity rating. The ratings are:

- Time required to implement: 68 minutes
- Is it currently being implemented: No
- Team familiarity: 4 team members

3.1.2 Summary

Approach	Time Required (Minutes)	Currently Implemented (Yes or No)	Team Familiarity (Ranging 1-4)
Using OpenLayers GUI and USGS's Astro JavaScript Libraries	137	Yes	2
Using Leaflets GUI and USGS's Astro JavaScript Libraries	113	No	3
Using Leaflets GUI	68	No	4

Table 1: GUI Layout Analysis Results

After researching the three different approaches to the GUI layout, the team chose the second approach as the solution. As seen in Table 1, implementing the example for approach number two was not the shortest, but implementing the final product will be shorter overall because USGS's Astro Libraries have the projection buttons and longitude and latitude menus already created. The GUI layout is not currently being implemented by USGS, but the team felt more comfortable creating the menus using Leaflet's GUI tools rather than using OpenLayers' GUI tools. For these reasons, the second approach was selected to be the chosen solution, but may require future design changes. Without being able to measure user familiarity and ease of use objectively, the team will test and update the layout as needed.

3.2 Projection Algorithms

There are many different well-developed mapping applications, including OpenLayers, Leaflet, D3, and Cesium. However, the main problem with these applications is that they only support Earth mapping projections: EPSG3857 and EPSG4326. This makes it incredibly difficult for USGS to map planetary bodies with data they have collected. For example, USGS has an implementation of OpenLayers, but they had to write all of the transformation functions for the projections. If they had not created these functions, the map would be incorrect or never display to the screen. A problem with writing these transformation functions is that USGS would have to rewrite them for every mapping application they want to support and use. The team's goal is to create a package that abstracts the projections so that they can be reused in different mapping applications without having to rewrite the code. Right now, the team's main concern is to make the package work with Leaflet and OpenLayers, then the team can move onto supporting other applications.

The important factors when considering how the team is going to package the projection algorithms are:

- Date of Last Update
- Amount of Examples: How many examples are on the application's main website
- Works with USGS Data in Leaflet: Yes/no/needs more testing
- Planetary Science Standard: Yes/no
- Portable: Yes/no/needs more testing

These metrics allow the team to make a decision with different types of solutions in mind. The first three metrics relate to how the team's package will be used in Leaflet. Because projections are difficult to understand, and combining them with virtual maps makes it harder, being able to follow along with examples online will be extremely helpful; therefore, the team will be seeing how many examples are on the application's website. If the team needs to use any third-party plugins, the team needs to take into consideration the last time they were updated since these plugins may be outdated and not work with Leaflet. The team also needs to make sure that Leaflet is able to view the maps correctly with the projections and data from USGS using the solution. The next two metrics rank the package in terms of all mapping applications it may be used in, including Leaflet and OpenLayers. Portability is the most important metric as it tells the team if it can be used across these different applications. We include “Needs more testing” as a choice for portability because there may be solutions that the team does not know work with all mapping applications. The next metric tell the team if the solution is already a standard in the planetary science community. Because this is an open-source project for USGS, the team needs to follow along with standards from the planetary science community as closely as it can. With these metrics in mind, the team can now analyze a few approaches to decide on the best solution.

3.2.1 Potential Solutions

There are three main approaches to implementing this package. As a proof of concept and since the project is going to be using Leaflet, the team will rate the approaches based off of how the package will be used in Leaflet. The team will still be considering how the package will be used in other applications by rating portability. The first two solutions utilize an already well-developed and supported package called Proj4js. Proj4js is a JavaScript implementation of PROJ; a library first developed by USGS but now maintained by the OSGeo Community. It allows users to define transformations from one projection to another with ease and is used in many applications developed by USGS. The third solution uses the already-defined transformation functions USGS has in their OpenLayers code. All solutions will be used in Leaflet by creating an instance of the Leaflet class CRS, based off of coordinate reference system, that tells the map what projection to use. How the team actually defines the transformations for these projections is going to be different.

1. Proj4Leaflet

The first Proj4js solution will use a Leaflet plugin called Proj4Leaflet, a package allowing users to define projections using PROJ and use them in Leaflet. Users can instantiate a CRS object by defining a projection with a proj-string. Then, users can call a method named “transform” to transform coordinates from one projection to another.

Since PROJ handles all transformations, the team does not need to write any transformation functions. The package will then define proj-strings to be used in the “transform” function calls. Then, in Leaflet, the team will only need to get the proj-string from the package to create a projection for the map the team is trying to view.

In order to test this approach, the team created a simple Leaflet map, defined a new CRS with the Proj4Leaflet plugin, and passed the CRS object into the map instantiation. The team used a simple map of Earth to project. The solution worked and the team was able to view the map. The team followed along with 1 of the 6 examples on Proj4Leaflet’s website [5]. It was not difficult to follow along with this example; however, the team needs to make sure the USGS data can be used with it. Since this solution uses PROJ, it is a standard and is portable across many mapping applications because most applications should have some sort of PROJ port.

After looking at Proj4Leaflet's GitHub repository, the team discovered that their last release was in January 2017. This may lead to problems since Leaflet's last update was in May 2019; however, after looking at Leaflet's change logs for their updates, the CRS class has not had any breaking changes added to it since 2017 [4]. After these tests, we can now rate this solution:

- Date of Last Update: January 2017
- Amount of Examples: 6
- Works with USGS Data in Leaflet: Needs more testing
- A Standard: Yes
- Portable: Yes

2. Create Leaflet Plugin

The second PROJ solution would require the team to create a new plugin that does the same thing as Proj4Leaflet. OpenLayers uses PROJ directly and the team could mimic their solution.

It is difficult to test this solution since it requires the team to create a new plugin, but the team can see how many examples are on Leaflet's website for extending their classes. There are two examples on how to do so [2][3]. If the first solution works with the USGS data, then this solution should since they will both be using PROJ for the projections. In addition, since this solution uses PROJ, a standard is being used to do the math for the transformations and it will be portable across multiple mapping applications (for the same reasons as the first solution). Because the team is creating a new plugin, the team will not be considering the date of late update. With these tests and considerations in mind, here are the ratings:

- Date of Last Update: N/A
- Amount of Examples: 2
- Works with USGS Data in Leaflet: Needs more testing
- A Standard: Yes
- Portable: Yes

3. Use USGS Transformation Functions

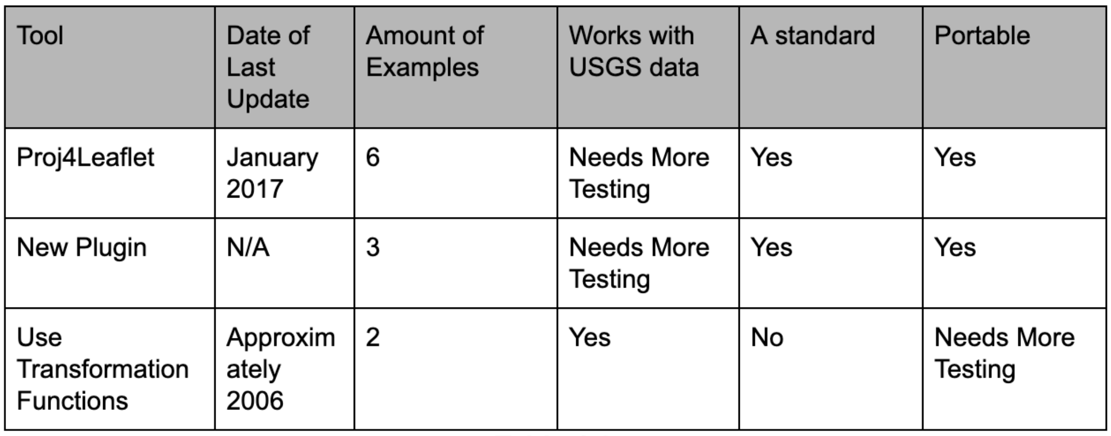
The third solution utilizes the transformations already defined by USGS in their OpenLayers code and abstracts the functions into a package that can be called by other mapping applications in different ways depending on the application itself. In order to use the package in Leaflet, the team would have to create new subclasses of CRS that define the projections needed. These classes will then have project and unproject functions that will call some functions in the team's package to do the transformations, i.e., PROJ will not be used to do the transformations. As for using the team's package in other mapping applications, this depends on how their projection class is created. The team knows that this solution will work with OpenLayers because that is what the current implementation uses and should work with Leaflet if a CRS subclass is created correctly.

In order to test this solution, the team tried extending the Leaflet CRS class by creating a cylindrical projection. The team did not have to write “project” and “unproject” functions for this example since cylindrical follows the already supported EPSG:4326 projection. The team followed along with this example from Leaflet [1] and the two documents on how to extend Leaflet’s classes from above.

The team’s main concern with this solution is that the USGS transformation functions are not up-to-date (since they were written 13 years ago). This requires extensive testing and research. Since the team is not using PROJ, this does not follow any standards by the planetary science community and it is difficult to know if the package is portable across all mapping applications. On the other hand, because USGS has used these functions before, the team knows that the USGS data will work with this solution. With these test and considerations in mind, here are the ratings:

- Date of last update: Approximately 2006
- Amount of examples: 3
- Works with USGS data in leaflet: Yes
- A standard: No
- Portable: Needs more testing

3.2.2 Summary



Tool	Date of Last Update	Amount of Examples	Works with USGS data	A standard	Portable
Proj4Leaflet	January 2017	6	Needs More Testing	Yes	Yes
New Plugin	N/A	3	Needs More Testing	Yes	Yes
Use Transformation Functions	Approximately 2006	2	Yes	No	Needs More Testing

Table 2: Projection Functions Analysis Results

Table 2 highlights all rankings from the 3 different tools the team could use for the projection package. It is difficult to pick a solution right now until the USGS team can test a few small examples to see what solution will work with their data. Ultimately, the team wants to try to use PROJ in the package since it is already well-developed and supported by the planetary science community. In addition, it will be very easy to do transformations between projections with PROJ. As for how to integrate the package with Leaflet, the team wants to try to go with the first approach since Proj4Leaflet is already developed and is referenced on Leaflet's plugin page. However, if the Proj4Leaflet plugin is outdated or does not work with the USGS data, then the third approach will be the team's next pick because it already works with the USGS data and the team does not need to develop a new plugin for Leaflet.

To further test these solutions, the will be creating a few Leaflet maps with three different projections: north-polar stereographic, south-polar stereographic, and cylindrical. These maps will use planetary data from a JSON given to the team by USGS. The team will then give the maps to the clients to test that they are projecting correctly.

3.3 Coding Tools, Format, and Style

For a project of this scale, the implementation of the chosen solutions will require the entire team to produce and test code. To ensure the team is writing clean code, a unified coding style will need be adopted and followed by each member. Not only must the team produce readable, bug-free code, the official Leaflet plugin rules and requirements for style and format must be followed to earn the title "Official Leaflet Plugin". Lastly, browser-compatible code that works on all major web browsers is essential to provide a quality, final product to the researchers and scientists who will be using it on various platforms around the world.

There are many coding tools available to help satisfy these requirements. The team narrowed down the choices and tested tools to get a better understanding of what they had to offer. The following metrics were used to qualify potential solutions for use:

- Time since last update: 0+ years(s)/month(s)/day(s)
- Number of Google hits: # out of #
- Number of weekly downloads: 0+
- Team-wide synchronization?: Yes/no

It is important to determine whether or not a given tool is maintained to ensure that the final product has long term support. To find the time since the last update of a tool, the team used the search utility included with node package manager (npm). npm lists the number of years, months, or days since the most recent version of the tool was published. To get a sense of the implementation difficulty, we used Google to search the name of the tool and counted the number of useful results (documentation, tutorials and guides, and example usages) out of number of total results. To gain a clear sense of the tool's popularity among developers, we attained the number of weekly downloads from npm. This helps to verify the tool's quality and usability through community adoption. Lastly, Assessing whether the tool is able to be synchronized across the team involved determining whether one configuration file can be used by each team member rather than each individual needing to create their own local file or configure settings.

3.3.1 Potential Solutions

1. Prettier.js

Prettier.js is an opinionated code formatter that automatically reformats files upon saving. Prettier.js reads the user-provided configuration file for formatting specifications, copies the file contents, then re-writes it to the file, following the specifications provided. The configuration file is in JSON format for ease of use, with few options and a default configuration predefined for simplicity. This tool can be included in a package.json file shared via project repository and then easily downloaded to each team member's local environment.

- Time since last update: 5 months
- Number of Google hits: 5 out of 7
- Number of weekly downloads: 5,655,085
- Team-wide synchronization?: Yes

2. Text Editor Preferences Export and Share

Most text editors provide configuration preferences for formatting files of all kinds. They generally also provide a mechanism to import and export configuration preference files which can be shared team-wide. This would definitely fulfil the requirements the team has, but with some drawbacks. Setting up the preference file required significant effort and a deep knowledge of the editor. The same editor would need to be adopted by each team member, which could potentially disrupt the ability to code for any individual who is not familiar with it. In terms of measuring this option, there were too many text editors available to get an accurate and objective measurement.

- Time since last update: N/A
- Number of Google hits: N/A
- Number of weekly downloads: N/A
- Team-wide synchronization?: Yes

3. ESLint

ESLint is a pluggable, linting utility for JavaScript and JSX. Code linting is a type of static analysis used to find problematic patterns or code that does not adhere to style guidelines. These rules are developer-defined and can include the built-in rules that are designed to identify and reduce potentially problematic code. ESLint can be configured to run in parallel with Prettier.js to automatically reconfigure files in terms of the formatting and style. Problematic code cannot be automatically reformatted and would need to be fixed manually by the developer. The configuration is specified in the ".eslintrc" file in the project's directory.

Eslint-config-airbnb-base is a node package that defines an ESLint configuration file that can be included in the ".eslintrc" as the configuration to be followed. To be an official Leaflet plugin, the Airbnb configuration with slight modification is used to standardize formatting and style across all Leaflet plugins. We must use this configuration to be formally recognized by Leaflet.

Eslint-plugin-import is a node package that is a required peer dependency for eslint-config-airbnb-base to support the integration of Eslint plugins and import of configuration files. This will allow Eslint to find the plugins specified in the configuration file. Together, these tools provided a simple and effective means of team-wide best-practices enforcement and unsafe pattern checking.

- Time since last update: 12 days
- Number of Google hits: 5 out of 7
- Number of weekly downloads: 8,065,379
- Team-wide synchronization?: Yes

4. Babel

Babel is a toolchain used to convert ECMAScript 2015+ code into a backward-compatible version of JavaScript in current and older browsers or environments. Babel can transform syntax, polyfill features that are missing in the target environment, be debugged, and is extremely compact. Ensuring code compatibility on all browsers and environments is easily accomplished using this tool. Babel uses a configuration file, ".babelrc", which can be configured to accept preset configurations made by the community.

Babel-preset-airbnb is a preset configuration package for Babel which includes the necessary options for Airbnb's style guide. This provides an easy way to specify the configuration automatically to Babel without having to manually specify the options.

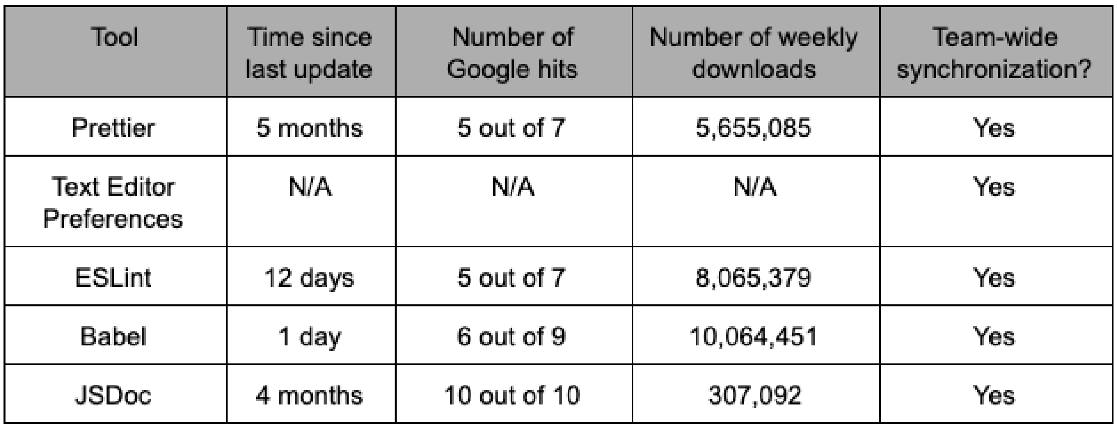
- Time since last update: 1 day
- Number of Google hits: 6 out of 9
- Number of weekly downloads: 10,064,451
- Team-wide synchronization?: Yes

5. JSDoc

JSDoc is an API documentation generator for JavaScript. This tool allows for documentation directly in our source code via block comments which are scanned by the tool to generate an HTML documentation website [9]. This provides simple and automatic documentation without the hassle of doing it manually, allotting more time for other non-trivial tasks.

- Time since last update: 4 months
- Number of Google hits: 10 out of 10
- Number of weekly downloads: 307,092
- Team-wide synchronization?: Yes

3.3.2 Summary



Tool	Time since last update	Number of Google hits	Number of weekly downloads	Team-wide synchronization?
Prettier	5 months	5 out of 7	5,655,085	Yes
Text Editor Preferences	N/A	N/A	N/A	Yes
ESLint	12 days	5 out of 7	8,065,379	Yes
Babel	1 day	6 out of 9	10,064,451	Yes
JSDoc	4 months	10 out of 10	307,092	Yes

Table 3: Coding Standards Analysis Results

Table 3 highlights the rankings from the tools the team tested. As the project is still in the early phases, the chosen solutions may change over the course of project implementation, but, the team has concluded that the following tools we be used:

- Prettier
- ESLint (with the mentioned configuration packages)
- Babel (with the mentioned configuration packages)
- JSDoc

Synchronizing text editor preferences across the team proved to be too complicated and hard to set up, as well as not allowing for flexibility in development environments for each developer. Considering that these tools have been used by thousands of developers around the world, CartoCosmos is confident that the chosen solutions will work well for the project.

To further test these tools, the plan is to develop a “Technology Demo” that encompasses the chosen tools to create a working sample of our end product. To use these tools for the demo, the team will create a ”package.json” file that specifies the packages we are using which can be downloaded with the command “npm install” from the terminal, while in the project directory. Each tool has a different configuration file that specifies what options to use and will need a small initial setup. Instead of writing out all of the options manually, the team will create the configuration file and point it to the Airbnb packages that are installed and each tool will locate it automatically.

There are many helpful tools when it comes to web development, team-wide synchronization, and mapping applications. Choosing proven tools to synchronize the team’s code format, style, and ensure the stability of the team’s code allows more time to be spent focusing on the overall implementation of the project. Reducing the time spent on trivial tasks like documentation and style enforcement will undoubtedly result in a better product for the client.

3.4 Search with Auto-Complete

For one of the stretch goals, the team has to implement a standalone auto-complete into a search function that will allow the user to enter partial values/names for planetary Nomenclature into a text box. This text box will then search the USGS web catalog and retrieve a list of values the user can select from without having to fully enter them. The problem with the current solution is that it does not allow for this and requires the user to click the search button and be taken to another page before being shown the retrieved values. The team's solution will implement an auto-complete feature for when the user is typing. The auto-complete will pull up a list of items attached to the text box that start with the value that the user has currently entered. The user can then click or use the arrow keys to select the item they want and be taken to its page in the web catalog, without performing a full search.

The team will be using the following metrics to calculate which solution is the best:

- Time since last update: Time Period
- Developer API existence?: Yes/no
- Speed of results list return: Time in ms.
- Fulfils intended project requirements?: Yes/no

3.4.1 Potential Solutions

The team needs time since last update and developer API existence in order to determine how much time the team would need for this stretch goal. Time since last update shows how recent the documentation is,. Developer API Existence shows if the developer has released comprehensive API for the program. Since this will take place towards the end of the project, the team will want to complete this as fast as possible. Speed is important as most researchers will know what they are doing and what they are looking for, so the team does not need to value descriptive results as much. This is because the researchers will want to get the data as fast as possible. Finally, the team needs the auto-complete to meet all the project requirements and fix the problem.

1. autoComplete.js

The first solution, autoComplete.js, is one of the most lightweight and fast solutions. autoComplete.js allows the team to input a list of values which can then be searched from using a textbox. It has zero dependencies which helps with usage and installation. It supports Microsoft Edge, Google Chrome, Firefox, Safari, and more, making it easy for many people on different browsers to use. autoComplete.js does not, however, support Ajax, which allows for synchronization between the browser and server [10]. This means that after the initial value load for the auto-complete, the retrieved list will not be updated with any values that are added afterwards to the server without a browser refresh. This would not be a problem for most users but could cause some delays while testing newly added nomenclature. Also, even though there is a decent amount of documentation on the functions, it is a bit hard to follow and is not as thorough as the auto-complete libraries for jQuery are. The team tested this with the example provided on the website. The data set seemed to include a wide set of values that autoComplete.js had to search through, it retrieved the data in a very short time, though.

- Time since last update: 5 Days
- Developer API existence?: Yes
- Speed of results list return: 113ms
- Fulfils intended project requirements?: Yes

2. jQuery UI Autocomplete

JQuery UI Autocomplete is one of the most in-depth tools the team could use, but this also leads to longer waiting times. It depends on jQuery to run which will complicate the installation process a bit more as the team would have to install jQuery UI on the client's system alongside the auto-complete widget. The jQuery UI Autocomplete has a lot of thorough documentation on how to use it and what each method does.jQuery UI Autocomplete's functionality encompasses all of autoComplete.js and includes more; it allows for keyboard input through jQuery events and system responses to certain items selected on the results list [13]. Because of its implementation with jQuery, the team will also be able to use jQuery methods and events with this auto-complete. This would, however, take a good amount of time as the team would have to learn the jQuery methods alongside the ones in the auto-complete widget in order to make the best use of the tool. JQuery is supported on all browsers, which is important for a tool used by many different potential users. As a final note, jQuery works alongside Ajax, which can be used to dynamically update the results list from the server. This will ultimately slow down the process of getting items. The team tested this one with a time test of the example given. The example did return the results list decently fast but not as fast as it could.

- Time since last update: 3 Years
- Developer API existence?: Yes
- Speed of results list return: 346ms
- Fulfils intended project requirements?: Yes

3. EasyAutoComplete

EasyAutocomplete is another jQuery using library that allows for Ajax but is more independent than the jQuery UI Autocomplete. Instead of having to learn jQuery as a whole, EasyAutocomplete provides a guide you can work through that includes all of the EasyAutocomplete specific items and methods and gives examples [11]. This guide also references commonly used methods from jQuery, which will trim down the learning time. It also offers the ability to edit templates to allow things like images in the return results, and uses categories the user can look up. EasyAutocomplete would require more user training to use those features. Pulling back a list based off of a category would not be very useful as most categories the team can find to base off of on the USGS Planetary Nomenclature would pull back huge lists, defeating the point. The team tested this by using an example. It was moderately fast and ran a little faster the jQuery UI Autocomplete did.

- Time since last update: 1 Month
- Developer API Existence?: Yes
- Speed of Results List Return?: 325ms
- Fulfils Intended Project Requirements?: Yes

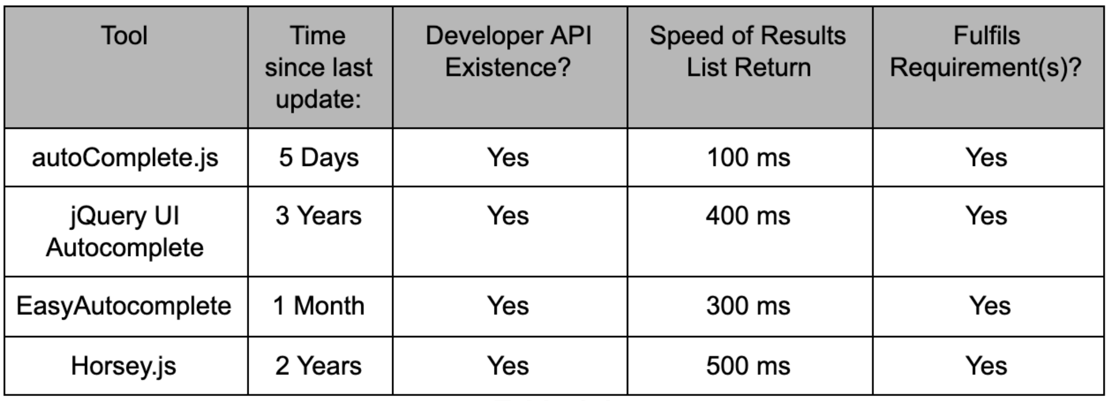
4. Horsey.js

Horsey.js is an auto-complete node package module that does not use jQuery or Ajax, but is more advanced than autoComplete.js [12]. Horsey.js allows for the use of key-value pairs, allowing users to look up a piece of text associated with a result and get the result. Horsey.js also allows for images in the results and the ability to use the textbox it is applied to as a drop-down list. Horsey.js is a great tool for getting more functionality than autoComplete.js while still not requiring jQuery knowledge.

Horsey.js was tested using the example above. It ran surprisingly slow, even with a small data set. Slower than the solutions that use jQuery.

- Time since last update: 2 Years
- Developer API existence?: Yes
- Speed of results list return: 500ms
- Fulfils intended project requirements?: Yes

3.4.2 Summary



Tool	Time since last update:	Developer API Existence?	Speed of Results List Return	Fulfils Requirement(s)?
autoComplete.js	5 Days	Yes	100 ms	Yes
jQuery UI Autocomplete	3 Years	Yes	400 ms	Yes
EasyAutocomplete	1 Month	Yes	300 ms	Yes
Horsey.js	2 Years	Yes	500 ms	Yes

Table 4: Auto-complete Tools Analysis Results

AutoComplete.js is the team's chosen solution. Although the other items offered more functionality, autoComplete.js is the simplest to use. The methods provided by the jQuery widget and plugin were nice but not needed, and would have required the team to put in a lot of time to learn jQuery for not many methods that would actually be used. Horsey.js would have been the same way. It would have decreased speed by much less but would have been over-complicating the problem with all of its methods. The team is prioritizing speed above all else, since most of the users will be researchers who already know what they want. The team just wants to speed up the process of them getting to it, not offering images or other fluff. The methods will be harder to learn but there are not that many of them, and once the team experiments and learns exactly how they work, they will not be a problem. If the client ends up wanting the extra functionality and does not mind the loss of speed and project time, the team will implement jQuery UI Autocomplete.

In order to prove the feasibility of autoComplete.js, the team will have to build a very basic example, measure its time, and compare it to the other possible solutions. Speed is what matters most here, so that will be the team's main measurement tool.

The team has many options, like the ones mentioned above, that have differing levels of functionality and ability. However, for the purpose of this project, less is more, meaning that the simpler and faster the auto-complete, the better it will be for both the team as developers and the client as a user.

4 Technology Integration

Each technological challenge and corresponding solution serves a crucial role in the overall success of the project. While choosing the optimal solutions, it is necessary to verify that each individual solution can be unified into a single, all-encompassing package that satisfies project requirements.

Since the technological challenges are specific and generally unrelated on a technical level, their integration does not require the team to follow strict guidelines. The GUI layout, projection algorithms, and coding tools will all be implemented as one package but have their own independent means of implementation, while the search with auto-complete feature is completely separate. To tie them together, the team will need to build a library extension of Leaflet to define the GUI layout, use Proj4Leaflet to convert existing projection algorithms into the Leaflet extension, and nest the team's coding tools as developer dependencies in the package.json with applicable configuration files in the main directory. Following the main package implementation, the team will be separately implementing a search with auto-complete features using autocomplete.js that will no be directly tied to the GUI or Leaflet.

The GUI layout will be implemented through Leaflet's provided GUI with tweaks made to include many visual elements from the current USGS solution that is currently in use. To accomplish this, the team will be adding embedded functionality onto the interface Leaflet provides and use CSS and JavaScript methods to make it visually appealing.

The projection algorithms, if the team is able to use the library Proj4Leaflet, will be as simple as importing the functionality and using its predefined functions to map the vector/raster data provided. If the team is unable to use the library with USGS data, they will be implementing the projections used by USGS in their current solution into an attached library. Either way, this functionality does not tie into the other solutions directly and the formulas are pre-derived for the team's use.

The chosen coding tools will be the first files in the project shell, using the aforementioned files with the configuration contents as shown. Once the package.json and other required configuration files are set up in the empty project shell, they will be pushed to GitHub so that the team can synchronize tools and begin implementation. After this initial setup is complete, there will be no further action required from anyone to maintain the tools.

Lastly, the search with auto-complete feature will be implemented as a separate package for use on the same web page as the planetary mapping application. It must search through a Web Feature Service (WFS) full of features corresponding to the body currently being viewed and open another web page where information on that feature is located. As this is separate from the mapping application and only needs to be able to read the current body being viewed from the web page's URL, there is no integration needed.

With this integration plan in mind, there are other challenges that arise that the team does not have control over. Because USGS is using MapServer, an open-source platform for publishing spatial data, to host their planetary data and MapServer only supports Earth projections, the team can only query it with Earth projection codes. This requires the team to give the projections the wrong code in the projection algorithms package, which may be misleading to people who read the code. In addition, the team has to make sure that parts of the GUI such as the scale bar are correct for the projections.

5 Conclusion

In conclusion, in order to allow researchers to continue making maps via virtual mapping applications, USGS's current implementation needs to be replaced. Without data accuracy and modularity, this problem will never be fixed. Throughout this feasibility document, the team has identified the key technical challenges that they are going to face while trying to implement the design. Having to use technologies like Leaflet and Jupyter Notebook, the team knows that they need to code for an open-source project and in JavaScript. In Table 5 you can see the technical challenges the team identified and approaches chosen to solve them. Overall, the team is confident and excited that these chosen solutions will allow them to create a vastly improved virtual planetary mapping application for USGS, NASA, and the planetary community. The next step for the team is to start working on a small example of a Leaflet map of Mars to show to the clients.

Technical Challenge	Solution	Confidence Level
GUI Layout	USGS astro with Leaflets GUI tools	Medium
Auto-Complete	autoComplete.js	High
Projection Algorithms	Proj4Leaflet	High
Coding Tools	Prettier, ESLint, Babel, JSDoc	High

Table 5: Summary of Decisions

6 References

1. https://leafletjs.com/examples/crs-simple/crs-simple.html
2. https://leafletjs.com/examples/extending/extending-1-classes.html
3. https://leafletjs.com/examples/extending/extending-3-controls.html
4. https://github.com/Leaflet/Leaflet/blob/master/CHANGELOG.md
5. https://github.com/kartena/Proj4Leaflet/blob/master/examples/wms/script.js
6. https://prettier.io/docs/en/index.html
7. https://babeljs.io/docs/en/
8. https://eslint.org/
9. https://devdocs.io/jsdoc/
10. https://tarekraafat.github.io/autoComplete.js//?id=introduction
11. http://easyautocomplete.com/guide
12. https://github.com/bevacqua/horsey
13. https://jqueryui.com/autocomplete/