Requirements Specification

Version 2

12/13/19

CartoCosmos

Scott Ames, Jacob Kaufman, Kaitlyn Lee, Christopher Moore

Sponsor:

USGS Astrogeology Science Center

Mentor:

Isaac Shaffer

Accepted as baseline requirements for the project:

For the client: _____ 12/12/19 *For the team:* _____ 12/12/19
Signature *Date* *Signature* *Date*

_____ 12/12/19
Signature *Date*

Contents

1 Introduction

An important part of space exploration is launching satellites into space to orbit planetary bodies, collect large amounts of data, and take images of these bodies. The data and images are sent back down to Earth and used by the planetary science community to create planetary maps. These maps play a crucial role understanding more about these bodies and their surfaces.

Figure 1: Mars Odyssey Map Highlighting Gale Crater

Today, all eyes are set on Mars since it 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. See Figure 1 for a map of Mars created using the Odyssey data collected. Without this orbiter and the system of maps it has gifted us, we would not be able to learn more about Mars and how we could land on its surface one day.

The Mars Odyssey mission is just one example of all missions that are carried out. Because there are so many planetary bodies to explore, the planetary science community is very large and consists of scientists, students, and developers from different organizations spanning the globe. Some of these organizations include the United States Geological Survey (USGS), the National Aeronautics and Space Administration (NASA) and 10+ mission teams operating under NASA, other countries' space agencies like the European Space Agency (ESA) and the Japan Aerospace Exploration Agency (JAXA), and 30+ universities. All of these entities play an important role in the research process.

Our client is a small team from the USGS consisting of Trent Hare and Scott Akins. Trent Hare is a cartographer and also works with other companies to try to get them to support the use of planetary data in their programs. Scott Akins is an IT specialist who focuses on web development and databases. They both work for the Astrogeology Science Center (ASC), a subbranch of the USGS, in Flagstaff, AZ.

The ASC consists of developers, students, and scientists all working together to conduct research. Currently, there are approximately 20+ scientists, 20 mission-support/web developers, 20 IT specialists, and 10 students working for the ASC. In the rest of this paper, we will be focusing on the mission-support developers and refer to them as just developers. The ASC is contracted by NASA to support the planetary science community, and they do so in different ways. The developers' main jobs are to create programs that aid in the map-making process and house the data collected during missions so that scientists do not have to store terabytes upon terabytes of data on their own machines.

The developers maintain the Integrated Software for Imagers and Spectrometers (ISIS) [1], a package used to manipulate and process images. They also develop other web-based tools for scientists to use. One job of the scientists is to use ISIS to create maps and work alongside planetary missions to conduct important research that would change the future of space exploration forever. When creating these maps, scientists use projections to portray these round bodies on the map's flat surface. Projections define the coordinate system to be used by the map, the bounds or extent of the map, and the center of the map. The World Geodetic System (WGS) defines one of the most-used projections for Earth . The current version of the definition is WGS84 [2], also identified as EPSG:4326. For more information about projections, see this handy guide created by the USGS [3].

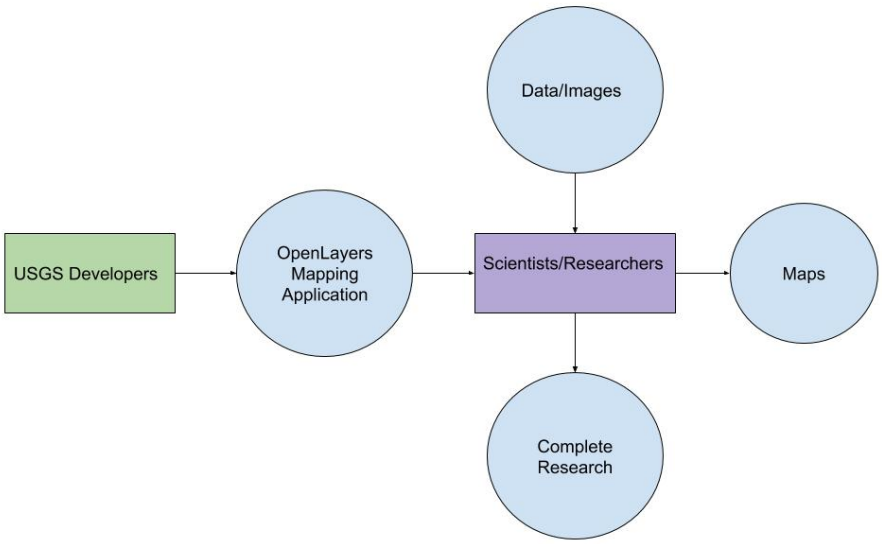

Figure 2: USGS Workflow

[1] https://isis.astrogeology.usgs.gov/
[2] https://en.wikipedia.org/wiki/World_Geodetic_System
[3] https://store.usgs.gov/assets/mod/storefiles/PDF/16573.pdf

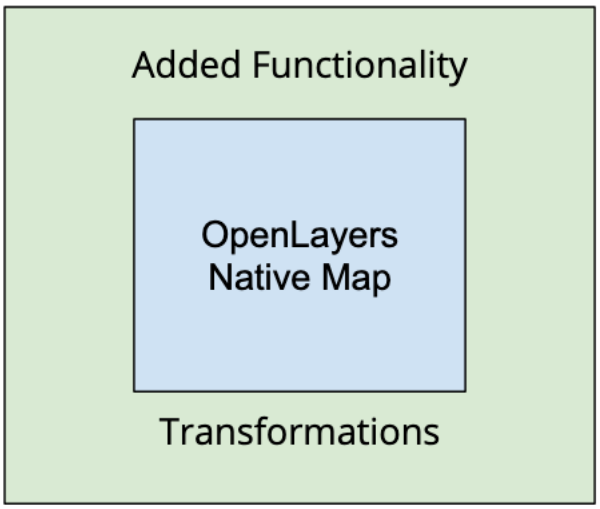

Figure 3: USGS OpenLayers Architecture

2 Problem Statement

One workflow for both the developers and scientists when creating maps and doing research is outlined in Figure 2. The scientists and researchers use the data collected during the missions to create maps of planetary bodies. The developers created a map-viewing application using the open-source package OpenLayers (OL) that displays dynamic maps. The scientists then use the OL implementation to view the maps virtually and are able to finish their research. The problem with the current USGS workflow can be broken down into two categories: (1) how it affects the developers, and (2) how it affects the scientists.

2.1 Problems for Developers

There are a few problems with the USGS OL implementation that are causing roadblocks for the ASC development team. The problems are:

- It is 13 years old.
- It is not modular.
- The development team had to write their own math packages.

Because the implementation is 13 years and not modular, it is difficult for the development team to update it. Further, their implementation uses OL version 2 released in June 2006. The current version of OL is 6 and it was just released at the beginning of November 2019. The architecture for the implementation can be seen in Figure 3. They created a wrapper around the OL native map object. The wrapper includes the added functionality that they need, but written using OL code (we will go further into the added functionality in the requirements section).

This architecture is not modular because if the USGS wanted to create more map-viewing applications, and they do, they will need to rewrite the code implementing the added functionality since it is written in OL code.

2.2 Problems for Scientists

It is very important that the USGS scientists, and other scientists that use the map-viewing applications, have an application that gives them all the tools they need to conduct their research. Currently, there are many map-viewing applications on the Internet that scientists may use but there are a few fundamental problems with them:

- They only support maps of Earth.
- They do not allow users to change their latitude and longitude settings.
- They do not allow users to change what projection the current map is in.

Because the main focus of these scientists is to research planetary bodies other than Earth, the first bullet is a critical issue. The next two bullets highlight the added functionality the USGS OL implementation created in their wrapper in order to support the scientists. Additionally, there is one other problem not caused by the mapping applications; there is no way for scientists to search for features on a body when they are doing research.

2.3 Conclusion

Finally, there is one other problem for both the scientists and the developers. The developers may update their OL implementation, but there is still just one map-viewing application scientists can use to complete their research. The USGS wants to be able to support more scientists by creating more map-viewing applications that will have the added functionality. Our solution explained below aims to solve both the problems of the developers and scientists.

3 Solution Vision

To solve the aforementioned problems, we will be creating a new mapping tool for the USGS with the functionality that they require. Our solution will involve the following:

- Create a portable JavaScript node package that contains the transformation functions needed to use projections other than Earth-based ones.
- Use Leaflet, another mapping tool, to create the maps.
- Create a brand-new GUI wrapping the Leaflet map.
- Update the code to make it more modular and usable for other mapping tools.

- Connect to the USGS GeoServer for the names of geographic markers for an auto-complete functionality.
- Build latitude and longitude switcher on map.

3.1 Creating a JavaScript Package

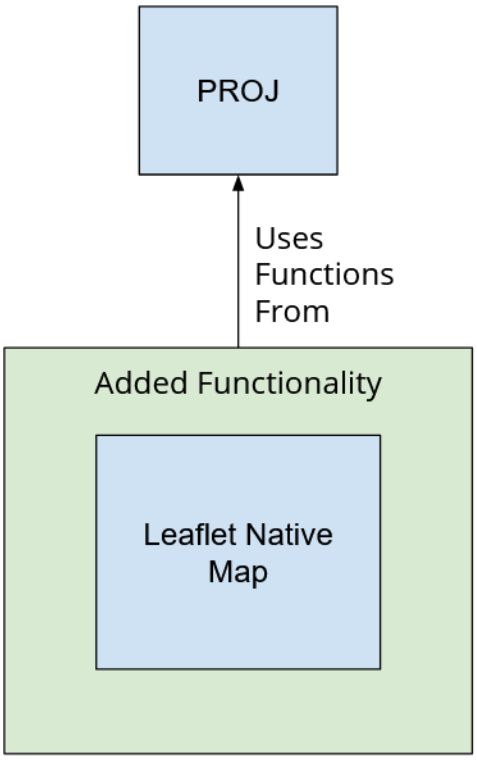

Figure 4: Leaflet with PROJ Functionality

The JavaScript node package we create will be responsible for making sure that Leaflet can be used to map planetary bodies other than Earth and contain the code the code for the back-end of the latitude and longitude switcher. This package will act as a plugin and be written for Leaflet. It will be as modular as possible because we want this code to be able to be used in any kind of mapping tool the USGS may start using in the future. We plan on using PROJ, an already built JavaScript library that defines the transformations we will need to use other projections. As shown in Figure 4, the PROJ functionality will be implemented and used by Leaflet instead of having the transformations inside of the Leaflet implementation. PROJ contains the transformations we will need to transform Earth data into other planetary bodies'. The plugin we create will be using these transformation functions to create the necessary projections for other planetary bodies. By using PROJ, we are using code that is already tested and proven to

work. It will severely cut down on development time since we do not have to write the transformations ourselves.

3.2 Using Leaflet

Leaflet is an open-source JavaScript library with built-in mapping tools. Leaflet does not come with as much functionality as OpenLayers but is able to better accept user modifications and plugins. This allows us to implement our code as a separate plugin that can be used for map-viewing applications other than Leaflet.

3.3 Creating a new GUI in Leaflet

Using Leaflet as our mapping tool allows us to update the old OL GUI to something more modern and helpful to the USGS. The new GUI will allow for the extra functionality needed by scientists such as the latitude and longitude switcher.

3.4 Connecting to the USGS GeoServer

We will be using the data in the USGS's GeoServer, an open-source server for sharing geospatial data, to auto-fill search data into partially filled searches scientists make to find features on a map. We will query GeoServer to get the requested geographic markers, such as craters or mountains, and allow the user to select and see the information on them. This will allow scientists to quickly find the data they are looking for and help them to remember if they forgot the full name.

4 Project Requirements

Our solution, which will aim to solve both the problems of the developers and scientists, will be two-part. The first part will be an interactive map, and the second a package with the needed transformations for projections and latitude/longitude coordinates.

In order to better understand our solution, we will discuss the requirements of the project. We will first go over the domain-level requirements, top-level requirements from the user's perspective. Then, we will further break down the domain-level requirements into functional, performance, and environmental requirements.

From a user-level perspective, we have identified three domain-level requirements that our project needs to satisfy:

- We must deliver an application that displays planetary data through an interactive viewer that is embedded in a web-based environment.
- The application must accomplish the primary goal of being research-driven in nature.

- We must create a portable package that can be used across multiple mapping applications.

To meet these rather broad domain-level requirements, we have broken them down in the following way.

4.1 Functional Requirements

The functional requirements discussed below dive deeper into how the interactive map will need to function from both the front and back-end, how our solution will aid in research, and how the portable transformation package will be broken up.

FR 1. Layer/Overlay Switcher

All Leaflet maps contain layers and overlays, but because the USGS supported target bodies (targets) have many different layers and overlays, we will need a menu to switch between them. Layers are the actual map and overlays are transparent layers that can be laid on top of maps. This requirement can be split up into sub-requirements in the following way:

- Display all available layers for a target planet.
  - Grab information on the requested target from the JSON given to us by the USGS.
  - Use the data from the JSON to query GeoServer to receive the Web Map Server (WMS) map layers (see ER. 5 Web Map Service/Web Feature Service standards).
  - Load the layers into Leaflet's native layer switcher.
  - All targets have a default layer that should be the first layer to display on the map.
- Display surface feature names tagged/symbolized by type for a target. Most of the bodies USGS have studied have named surface features. These names must be view-able as an overlay.
  - Grab and store map overlays based on the target planet and projection from the JSON given to us by the USGS.
  - Load the overlays into Leaflet's layer switcher.
- When a user changes the projection, the layer/overlay switcher should reload the layers associated with the requested projection.
- If a target does not have any layers with a requested projection, display a message that no map is supported.

FR 2. Projection Switcher

All Leaflet maps have a projection, but because Leaflet only supports Earth projections, new projections will need to be defined for the USGS supported targets.

- Define three projections for each target: cylindrical, north-polar stereographic (north-polar), and south-polar stereographic (south-polar).
  - Grab and store information about the projections from the JSON given to us by the USGS. This information includes the bounds and projection name (cylindrical, north-polar, or south-polar).
  - Correlate the projection name to a projection code in the following way:
    - cylindrical → EPSG:4326
    - north-polar → EPSG:32661
    - south-polar → EPSG:32761
  - Register the projections by their codes so that they can be used by the map.
- After the projections are created, set the map's projection to the newly created, requested projection by its code.
- When a map of the requested target is first loaded, it should load the layers with a cylindrical projection by default. All targets have at least one layer with a cylindrical projection.
- When a user changes the projection, it should reload the layers associated with the requested projection and set the native Leaflet map's projection to it so that the mouse coordinates and scale bar are properly displayed.

FR 3. Scale Bar

Each map will need to have a scale bar so that users can gauge the approximate distance between features. Because different targets have different radii, the scale bar will need to change based on the target's radii.

- Set scale bar based off of the projection of the map. The projection tells us the radii of the target and the center of the map.
- When a new layer is loaded, the scale bar should refresh and show the correct information for the new layer.
- When a user zooms, the scale bar should change based off of the zoom level.
- Units will be in kilometers or meters, depending on the zoom level.

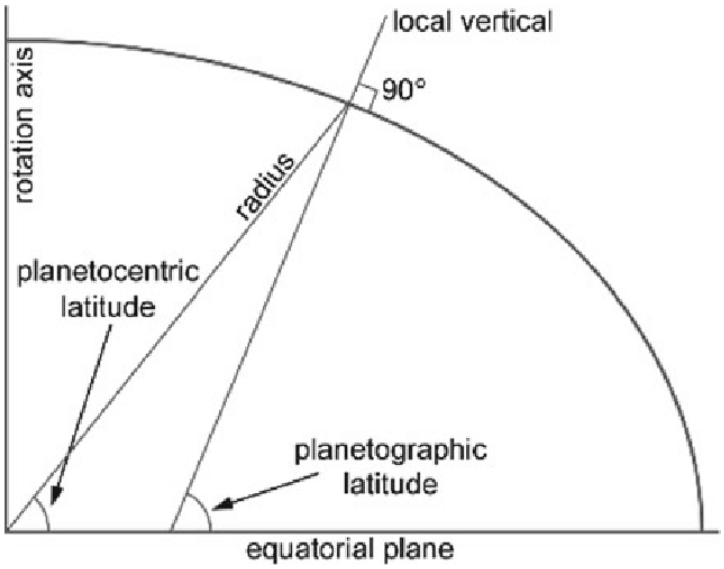

Figure 5: Planetocentric vs. Planetographic

FR 4. Latitude and Longitude Switcher

Leaflet's mouse position plugin allows users to see mouse position coordinates on any map, but since Leaflet only supports Earth maps, we can only see longitudes and latitudes native to Earth. New longitude and latitude ranges will need to be defined for the USGS supported targets. This requirement can be split up into different sub-requirements in the following ways:

- Earth's longitude range is recorded from $-180°$ to $180°$, while other planetary bodies are standardized on the $0°$ to $360°$ range. We must be able to report both on any planetary map.
  - Grab the mouse position coordinates for longitude and transform the native $-180°$ to $180°$ to $0°$ to $360°$ with a simple transformation.
  - Register the two longitude ranges in a menu allowing for any planetary map to have both ranges accessible, triggering on and off the longitude transformation.
- Planetary bodies come in various types of "round" and, depending on the shape, will require that their latitude be defined planetographicly or planetocentricly. Planetocentric is native to Leaflet because it is native to Earth. Planetographic is used to measure latitudes of ellipsoid/oblate bodies. Looking at Figure 5, we can see the difference in planetocentric and planetographic [4]. Planetocentric is the angle formed by connecting the center of the planet to a point on the surface of the planet. Planetographic, on the other hand, is the angle measured between a tangent line and the equatorial plane. We must be able to report planetocentric and planetographic on any planetary map.

[4]https://proj.org/operations/conversions/geoc.html

  - Grab the mouse position coordinate for latitude. Transform the latitude from planetocentric to planetographic because Leaflet defaults to planetocentric.
  - Register the two latitude ranges in a menu allowing for any planetary map to have both ranges accessible, triggering on and off the latitude transformation.

- Positive east is where longitude increases to the right. Reporting positive east longitudes is the default for Leaflet and Earth maps, but some planetary bodies are also reported in positive west longitudes (where the longitudes increase to the left).
  - Grab the mouse position coordinate for longitude.
    * If the longitude range is $-180°$ to $180°$ we must transform the longitude range to $18°0$ to $-180°$.
    * If the longitude range is $0°$ to $360°$ we must transform the longitude range to $360°$ to $0°$.
  - Register the positive east and west longitudes into a menu allowing for any planetary map to have both ranges accessible, triggering on and off the latitude transformations.

FR 5. Auto-complete

When completing research on a specific target, scientists need the ability to type in a partial value into a search and have the geographic markers with that partial value in their name appear as part of an auto-complete functionality.

- We will connect to the USGS GeoServer.
- We will get back the names of all the geographic markers on the planet that is currently being viewed.
- Scientists will begin typing in a feature name.
- We will then filter the geographic markers to one containing the partial value that was typed in.
- Suggest possible completions based on the list.

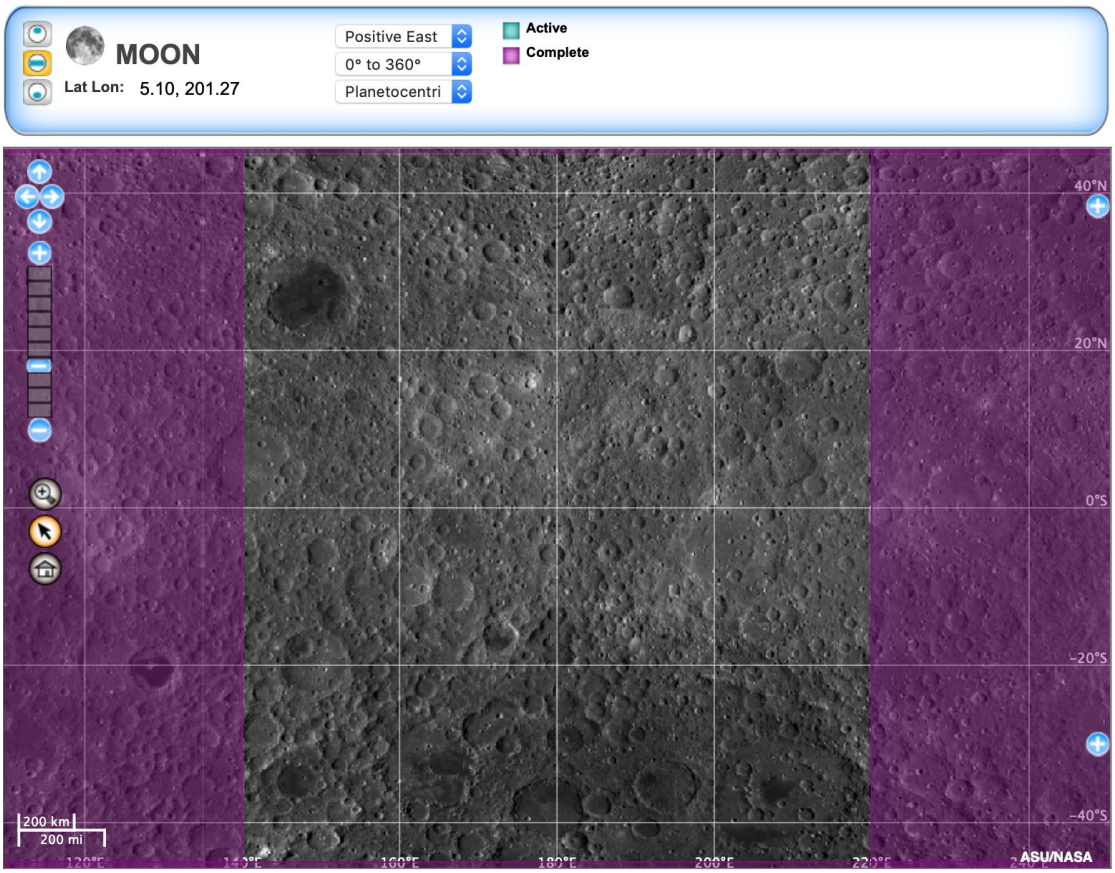

Figure 6: USGS's current OpenLayers map viewer with the moon as its target

FR 6. Graphical User Interface (GUI)

As the primary means of display and interaction with the user, the GUI is one of the most essential requirements for this project. The GUI contains both the map display itself and the menu that handles all user interaction. See Figure 6 for an example of the OL GUI that our solution will be based off of. As the front-end of our application, there are multiple requirements it must satisfy.

- Display the map data in an interactive way, allowing for map movement.
- Display the coordinates of the location underneath the cursor.
- Display the target body's name.
- Display a scale bar in km or m (see FR 3. Scale Bar).
- Provide a menu with mechanisms to switch between options for the target body:
  - All available layers, including surface feature names (see FR. 1 Layer Switcher).
  - 0° to 360° or −180° to 180° longitude ranges (see FR. 4 Latitude and Longitude Switcher).

- Planetocentric or planetographic latitude definitions (see FR. 4 Latitude and Longitude Switcher).
- North-polar, south-polar, or cylindrical projections (see FR 2. Projection Switcher).
- Zoom in and out of the current map, refreshing the resolution.

FR 7. Code Documentation

The main objective of this project is to replace and improve upon our client's current implementation. It suffers from poor documentation and file structure which negatively influences maintainability. To ensure our project is easily maintainable and can be used for many years to come, we will need to provide a well-documented codebase.

- Each class, function, class member variable, etc. must provide a clear description of its intended use and function.
- We must provide a web page with this information to easily view and navigate through it.

FR 8. User Manual

To ensure that our users can easily adopt a new and unfamiliar interface, we must provide a user manual for any users needing assistance. The user manual:

- Must be accessible from the web page holding our embedded mapping application or be included as internal functionality.
- Must provide beneficial instructions that successfully assist users of all skill levels to operate the application.

For example, this means that we can use a web page containing the instructions for the application's use or a menu/button that brings up the instructions. To determine what needs to be included in the user manual, we will examine how users interact with the viewer during user testing.

4.2 Performance Requirements

This product is being built for the planetary science community to aid research, and we must cater it to that use case. This means we need to not only guarantee the accuracy of the data being provided to them, but also test that our final product provides the usability and functionality that they require.

PR 1. Data Accuracy

To aid the research-driven goal of this project, there are multiple factors which must be considered to provide accurate data to our users.

- We must ensure that the application's scale bar and zoom feature is providing accurate data for each target (see FR 3. Scale Bar). We need to be as accurate as the OL implementation is, requiring us to compare all targets and projections in our Leaflet implementation to the OL implementation.
- The coordinates displayed from the location of the cursor must be accurate to the tenths place.
- The surface features displayed on the map must be positioned at the correct coordinates
- All latitude and longitude markers must be located at the correct position in regards to the currently selected options.

PR 2. User Testing

To ensure that we are able to provide our application to as many users as possible, we must test functionality and usability with our end users. The testing will be on the USGS scientists and consist of the following:

- Analyzing how long it takes until the user considers themselves familiar with the application.
- If the user is not familiar within 10-15 minutes of using the application, the test should be considered a failure.
- Surveying their experience using the application, e.g., scoring usability, layout, functionality, etc.
- Asking for requests of any additional features to be included in future updates.
- Examining user interaction to identify what needs to be included in the user manual (see FR 8. User Manual).

PR 3. Portability

The JavaScript package we create must be able to be used in multiple different mapping applications. Because only an OL and Leaflet implementation will be developed at this time, the package must be usable in both implementations. This requires all OL and Leaflet code to be extracted from the package. In the future, this package will be used in other mapping applications.

4.3 Environmental Requirements

Our clients have some specific constraints for this project that we must adhere to. These environmental requirements include what language our codebase is built with and specific compatibility with other software that we must implement. All of these requirements are specified to cater to specific use cases or general usability of our project.

ER 1. Official Leaflet Plugin

Our clients at USGS are hopeful that we can submit this project to Leaflet and become an official Leaflet plugin. To do this, we must adhere to some of Leaflet's requirements regarding documentation and file structure.

- We must follow Leaflet documentation standards.
- We must follow a file structure Leaflet has specified, involved a directory consisting of a root folder "App", then five sub-folders:
  - /src - JS source files
  - /dist - minified plugin JS, CSS, images
  - /spec - test files
  - /lib - any external libraries/plugins if necessary
  - /examples - HTML examples of plugin usage
  - README.md
  - LICENSE
  - package.json
- We must publish our project to node package manager for public use.

ER 2. Leaflet/JavaScript

One of the major requirements of the project is that we use Leaflet for all the base functionality and display of the maps. Leaflet was chosen by the the USGS for this project because it is extremely lightweight and developer-friendly. As Leaflet is a JavaScript framework, we also must use JavaScript for the application's codebase.

ER 3. Compatibility

As is standard with web applications, compatibility with all web browsers is extremely important to support inclusion of as many users as possible. This also means supporting external use of the application through other programs. In terms of this application, we must guarantee the following:

- The application must be compatible with all modern web browsers.
- The application must be able to interface with the Jupyter Notebook to facilitate sharing of data to support researchers and scientists.

ER 5. Web Map Service/Web Feature Service standards

Our project must follow the Web Map Service (WMS) and Web Feature Service (WFS) standards. This means that when we request map data through Leaflet, there is a specific format for the query requesting this information. These standards require us to:

- Follow general Hypertext Transfer Protocol (HTTP) request rules.
- Use the "GetMap" request using request parameters of:
  - VERSION=1.3.0 (mandatory): Request version
  - REQUEST=GetMap (mandatory): Request name
  - LAYERS=layerList (mandatory): Comma-separated list of one or more map layers
  - STYLES=styleList (mandatory): Comma-separated list of one rendering style per requested layer
  - CRS=namespace:identifier (mandatory): Coordinate reference system
  - BBOX=minx,miny,maxx,maxy (mandatory): Bounding box corners (lower left, upper right) in CRS units
  - WIDTH=outputWidth (mandatory): Width in pixels of map picture
  - HEIGHT=outputHeight (mandatory): Height in pixels of map picture
  - FORMAT=outputFormat (mandatory): Output format of map
  - TRANSPARENT=TRUE—FALSE (optional): Background transparency of map (default=FALSE)
  - BGCOLOR=colorValue (optional): Hexadecimal red-green-blue colour value for the background color (default=0xFFFFFF)
  - EXCEPTIONS=exceptionFormat (optional): The format in which exceptions are to be reported by the WMS (default=XML)
  - TIME=time (optional): Time value of layer desired
  - ELEVATION=elevation (optional): Elevation of layer desired

5 Potential Risks

With the development of any product, there are always potential risks that must be considered. Since our project must show an interactive Leaflet map of other planets on all modern internet browsers, there are inherent risks involved. These risks are:

- Leaflet no longer being supported.
- Leaflet's codebase updating.
- Browser updates.
- GeoServer crashing.

By analyzing these potential risks we can develop different mitigation solutions to help minimize them.

Leaflet No Longer Being Supported

Since Leaflet is an open-source application, there is no telling how long the Leaflet team is going to be supporting it. If Leaflet's team decides to no longer support Leaflet, it could potentially cause our planetary mapping application to not work on all modern browsers. Since Leaflet is written in JavaScript, it must be maintained and updated for browser compatibility. The Likelihood of Leaflet no longer being supported in the next few years is low. This is because the Leaflet team is very active online and have just released update 1.6.0 on November 17, 2019. If this does happen to occur during the implementation of our project, we will have to maintain Leaflet's JavaScript code ourselves. We could do this very easily, so the severity of this is low.

Leaflet's Codebase Updating

If Leaflet updates their codebase, it could cause our program to no longer function correctly. For example, if Leaflet changes the parameters it uses to initialize its maps, our planetary maps would no longer work. From looking at the Leaflet's release page [5], Leaflet roles out large updates roughly every six months with minor updates in between. Considering that Leaflet updated in early November, the probability of having a major update during the duration of this project is relatively moderate. If there is an update that happens to break our application, it would not be very severe because we are making our application modular. This means we will only need to update a very small amount of JavaScript code in order to get our application working again.

[5] https://github.com/Leaflet/Leaflet/releases

Browser Updates

Writing JavaScript so that it is cross-compatible for all modern browsers is very difficult. Each browser has different technical features and that includes JavaScript compilers. All JavaScript compilers are based on ECMAScript, a standardized version of JavaScript, but there are different versions of it. At this point in time, all browsers follow version three of ECMAScript and newer browsers use versions five and six. If a browser decides to update its JavaScript compiler to the newest version of ECMAScript, our mapping application may no longer function correctly. The chance of this happening is relatively high because browsers update all the time and modern browsers are making the move from version five of ECMAScript to version six. This issue is a very severe risk to our project because we can not change the compiler used for browsers. In order to manage this, we are going to be using an open-source program called Babel. Babel is a JavaScript compiler that takes in version six of ECMAScript and converts it into the browser-compatible JavaScript for the earlier versions of ECMAScript. This will allow our codebase to be compatible with all browsers, even if they update to the newest version of ECMAScript.

GeoServer Crashing

In order to get the data layers for our Leaflet planetary maps, we are sending specific calls to the USGS's GeoServer. The call is a URL that asks for a very specific layer of the target planet we are trying to map. When GeoServer receives this call, it sends back the layer if it has it. Leaflet then loads and displays the layer on the map. If a user is trying to access any of our planetary maps while GeoSever is down, no layers will be displayed. The likelihood of GeoServer crashing is moderate because there are a lot of factors that can cause a server to crash. Some factors that cause servers to crash daily are network issues, hardware issues, high server workload, and many others. The severity of this would be relatively small because even though our application would not be working, servers only usually crash and stay offline for a short period of time. Also, since the GeoServer we are using is the USGS's, we can count on them to keep it up and running for the use of our application.

Risk	Likelihood	Severity	Mitigation Solution
Leaflet No Longer Being Supported	Low	Moderate	Updating Leaflet's codebase ourselves
Leaflet's Codebase Updating	Moderate	Low	Modularity
Browser Updates	High	High	Babel
GeoServer Crashing	Moderate	Low	USGS

Figure 7: Risk Analysis Table

In conclusion, in order to allow our planetary mapping application to run on all modern browsers, we must consider the risks in Figure 7 shown below. We have taken into consideration the likelihood and severity of each risk. With the different mitigation solutions shown above, we hope to decrease the severity of these potential risk factors.

6 Project Plan

Our tentative plan for completing the planetary mapping application is split into two semesters, fall and spring. The fall semester is the planning phase of the project and the spring semester is the development phase. We are currently toward the end of the planning phase, as shown in Figure 8 in the appendix.

As seen in the diagram there are three major milestones remaining in the planning phase of our project. We have completed our technical feasibility study and are currently working on our technical demonstration and requirements acquisition. The requirements acquisition will be completed by the USGS approving the features of our virtual planetary mapping application with a signature. This will act as a contract for what we must implement in the implementation phase. The technical demonstration will show the key components of our projects and how they work together.

In the first two months of our spring semester shown in Figure 8, we will be implementing our planetary mapping application using Leaflet. Our plan for the implementation will be split into the map and the GUI class. The map class will contain the layer switcher, projection switcher, and the latitude and longitude transformations. The GUI class will contain different menus and buttons needed to use the technologies in the map class. These two classes will then be connected to Leaflet's codebase in order to display maps of other planets. This will complete the primary implementation of our planetary mapping application. As a stretch goal, we will then connect our application to the Jupyter Notebook to allow researchers to share maps of other planets. At this stage, we will assess how much time we have left to implement any added functionality into our mapping application. Once we complete the implementation of the stretch goals, we can finish our application by testing it. Testing our mapping application in-house and with the USGS researchers will ensure we deliver the best product to the USGS.

7 Conclusion

The planetary science community, which is very large and spans the globe, consists of both developers and scientists that work together to create maps. Without these maps, planetary missions would be impossible. Our clients, Trent Hare and Scott Akins, who work for the USGS, are developers and help in the map-making process. In order to do research with these maps, the USGS scientists need a mapping tool. Currently, the USGS has an OL implementation, but as discussed, there are many problems with it.

We have both developer and scientist problems that need to be solved. The problems for developers mostly pertain to how old the code is and the need to update the codebase. The problems for scientist show wanted features that the mapping applications do not have that are necessary for research.

In order to solve these problems, we will be creating a new mapping tool with Leaflet and a portable package. The new Leaflet tool will contain the extra functionality such as the latitude and longitude switcher, projection switcher, and scale bar. The package will contain the back-end code for the projection and latitude and longitude switchers. Our package will be portable and able to be used in other mapping tools such as OpenLayers. The USGS will be able to use this package in future implementations.

Our requirements are the project guidelines we have to follow in order to make sure our solution follows all of USGS's guidelines. The requirements fall into three categories: functional, performance, and environmental. Functional requirements outline how the users want to interact with Leaflet in a fast and efficient manner. These pertain to creating a projection switcher, adding a scale bar, updating the GUI, letting users switch the current layer and the latitude/longitude, and having an auto-complete functionality for searches. Performance requirements outline how fast, accurate, and easy to use Leaflet must be. We will accomplish this by having specific guidelines for data accuracy and user testing. Finally, environmental requirements outline the the main parts of our solution that are required to be us the USGS. This pertains to us writing accurate documentation for our code, building a user manual, using Leaflet for base functionality, making sure our solution is compatible with the Jupyter Notebook, and adhering to WMS and WFS standards.

Our project plan details how we are going to break up the project into smaller tasks and when we will have these tasks completed by. So far, we have completed the technical feasibility, requirements acquisition, and technical demos. These demos showcase a Leaflet map showing Mars' layers and projections, a Mars Leaflet map being creating in a Jupyter notebook, GUI layouts for future implementation, and documentation examples following Leaflet's standards. Next, we will start abstracting the Mars demo so that the Leaflet map can be instantiated with all of the USGS targets.

Because this project will go on the be used for years by organizations like the USGS and NASA, it is extremely important to us to make sure this project is the best it can be. We want to make sure that we are creating a solution with all the functionality scientists need in order to complete their research. We are confident that our code will be up to the task of being used in an official capacity by a government agency. Playing a role in space exploration is a tremendous honor, so we are extremely excited to complete this project and see it being used in planetary missions.

8 Appendix

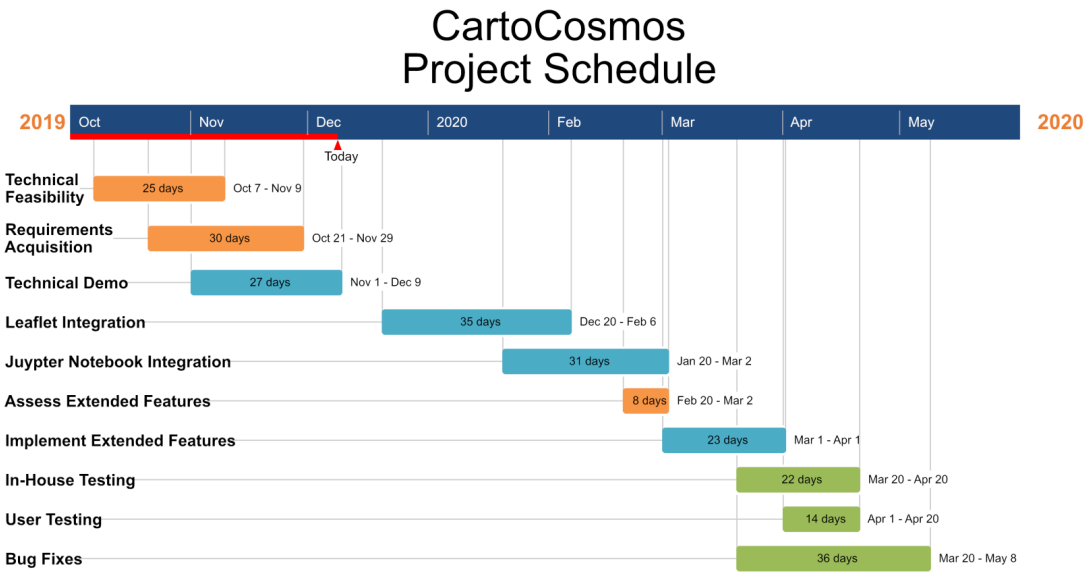

Figure 8: Schedule