Software Test Plan

4/01/2022

GeoSTAC

Sponsor:
United States Geological Survey (USGS)
Astrogeology Science Center

Mentor: Melissa Rose

Team Members:
Jacob Cain
Zachary Kaufman
Gavin Nelson
Amy Stamile

Version 1.0

Table of Contents

1 Introduction

The USGS Astrogeology Science Center serves data using a community developed standard to the planetary science community to access Analysis Ready Data (ARD). USGS has already distributed this analysis ready data through the Spatial Temporal Asset Catalog (STAC). STAC is a standard for storing, discovering, and analyzing spatial-temporal data to describe various geospatial information. This provides better indexing and discovering of the analysis ready data[1].

In 2019, the USGS assigned an NAU capstone team to develop an interactive web map that supports planetary data. The web map is called CartoCosmos, reflecting the capstone team's name. The CartoCosmos team developed a plugin extension for Leaflet, an open source Javascript library for interactive maps. This plugin extension was developed to support mapping of planetary data sets.[2] The CartoCosmos web map is great for visualizing large planetary image mosaics, but has no support for visualizing the individual STAC assets or STAC catalogs the clients wish to make available.

The overall vision for this project is to have the existing CartoCosmos web map display a certain number of footprints for specified planetary bodies. These footprints will be interactive: The user will be able to individually click a footprint and display the information associated with it. The user will have the option to view the cloud optimized geotiff image associated with the footprint. In addition, the user will be able to select multiple footprints. This will display a table of available footprints of the selected area and allow the user to select a footprint based on the table. Finally, the user will be able to use a filtering tool that will allow them to search within the existing rendered footprints to display only the footprints associated with the information searched.

The following sections discuss in detail how this project plans to be tested. In order to provide through testing, this document splits the testing into three categories:

- Unit Testing
- Integration Testing
- Usability Testing

The unit testing discusses how each function or method is being individually tested and testing metrics associated with each unit test. The integration testing section discusses the testing of integration between the web application and the STAC API. This goes into details of testing this integration through contract testing. The final

[1] https://stacspec.org/
[2] https://ceias.nau.edu/capstone/projects/CS/2020/CartoCosmos-S20/#/

section is usability testing. This section discusses in detail how testing will be administered to specific user groups and how each test will be measured.

2 Unit Testing

Unit testing is used to ensure that the code from individual functions of a piece of software are working correctly. Unit testing will help to detect and protect against any bugs that could arise in the future. A lot of unit tests are very simple that only require input and output verification. Although, other unit tests are used to test creation of objects and ensure the object contains the correct information. Since this project is built on top of an already pre existing capstone project, CartoCosmos there are already existing testing components for each of their separate modules (Leaflet, Jupyter Notebook, and Autocomplete). Since this project only updated the Leaflet module, this test plan focuses on updating that particular module. The plan is to create two different testing plans for inside the app, the React components and JavaScript components.

In order to test the newly updated Leaflet module, the following testing frameworks will be used: Jest, React Testing Library, and Mocha. This is to keep the testing components consistent with the existing testing frameworks that CartoCosmos used in their initial tests. Jest and React Testing Library will both be used to ensure the creation of our GUI components since it is written using React. Jest is a JavaScript test runner that allows access to the DOM using jsdom. React Testing Library is a set of helper functions that allows the user to test React components without relying on their implementation details. The main JavaScript code will be tested using Mocha. Mocha.js is an open-source JavaScript test framework that runs on Node.js and in the browser, it was designed for testing both synchronous and asynchronous code with a simple interface.

2.1 Astro

The Astro Module is responsible for the entire back-end functionality of the GeoSTAC planetary map. This module was originally created by CartoCosmos to call the USGS GeoServer, create the Leaflet map, and add support for changing projections and lat/lon settings. The team will be creating tests for only new classes and functions that we created inside of the already existing classes. The module contains a mixture of classes inherited from Leaflet classes and normal JavaScript classes and continues to follow CartoCosmos object-oriented paradigm. Every new class and function will have its own Mocha testing code. This will ensure every new class and function works in its intentional way.

2.1.1 AstroMap

AstroMap is the main class of the module and utilizes the other classes described below. It inherits functionality from L.Map and is used the same way as its parent class. The team updated the AstroMap class to load a layer of footprints gathered from the USGS STAC API onto the already existing CartoCosmos planetary map. This class imports functionality from the NPM staclayer and class ApiJsonCollection in order to get the data from STAC API and display it on planetary map. The Mocha unit test for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Loaded Footprint Layer	Test the creation of the footprint layer on the Leaflet Map	“Mars, ?page=1“ “MARS, ?page=1” “mars, ?page=1”	Instantiate a map with Mars and a single page of loaded footprints and an AstroMap object without errors.
Loads GeoTiff thumbnail from user click	Test the loading of GeoTiff thumbnail on Leaflet map	GeoJSON	Input is non-null object and loaded GeoTiff thumbnail on Leaflet with no objection errors

2.1.2 ApiJsonCollection

The ApiJsonCollection class handles the call to the STAC API to receive a JSON result in GeoJSON format and GeoTIFF assets to be displayed on the plantery map. The parameters inside the GeoJSON consists of URLs to the different GeoTIFF assets and metadata for the GeoTIFF. The class also handles storing and controlling the number of features that are displayed on the map using a pagination system. The Mocha unit tests for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Collect item collection from API call	Test the ability to call the STAC API and retrieve the JSON	“Mars, ?page=1“ “MARS, ?page=1” “mars, ?page=1”	An non empty JSON object
Set and retrieve the max number of pages possible	Test the ability to set and retrieve the max number of page possible	An array of features	An array of GeoJSON objects

Set and retrieve the value of the number of footprints	Test the ability to set and retrieve the value of the return number of footprints	An integer greater than 0	The same number as inputted: An integer greater than 0
Set and retrieve the max number of pages	Test the ability to set and retrieve the max number of pages possible	An integer greater than 0	The same number as inputted: An integer greater than 0
Set and retrieve the current page number showing	Test the ability to set and retrieve the current page number showing	An integer greater than 0	The same number as inputted: An integer greater than 0

2.1.3 AstroDrawControl

The Astro Draw Control class handles the back-end when a user draws on the Leaflet web map. Since the class inherits L.Control, it is added to the AstroMap in the same way as other controls, like the zoom control. The class also handles STAC query searching and auto populating the query box on the app GUI. The Mocha unit test for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Add draw controls to the provided map	Test the ability to add the control to AstroMap object	AstroMap Object	An AstroMap object with Layer Control attached with no errors.
Renders all footprints inside user drawn shape	Test the ability to load the footprints that intersect the drawn area	A string with the drawn shape's coordinates	A non null String object

2.1.4 GeoTiffViewer

The GeoTIFF Viewer class is used to display a pop-up with a given GeoTIFF assets thumbnail and meta-data displayed to the user. The class creates a modal object inside of the app's HTML div tag and then moves the blurs out the planetary map in the background. The GeoTIFF asset thumbnail is updated when a user clicks on a different footprint from the search results container. The Mocha unit test for this class are described below:

Unit Test	**Purpose**	**Sample Input**	**Expected Output**
Creates the	Test the ability to	String object with	An instantiate modal and

modal pop-up inside the app div	create a modal pop-up for the GeoTIFF viewer	the name of the image div	the methods return true
Display the GeoTIFF thumbnail to the user	Test the ability to display a given GeoTIFF url to the user	String with a given GeoTIFF thumbnail URL	Non-null DOM object

3 Integration Testing

Integration testing is a vital key part in this testing plan. Integration testing will ensure that every individual component of the project is working as intended and each module is interacting with and delivering the correct data to and from other modules. Along with being able to ensure that the modules and datasets are functioning properly, the team will also test and monitor the application's interactions with the STAC API. As for the implementation plan for integration testing, the plan is to build off of the existing testing framework that CartoCosmos provides. This will allow the team to utilize the powerful testing frameworks that CartoCosmos chose such as Mocha.js. Along with building off of the existing testing frameworks that CartoCosmos has provided, the team will be implementing Contract Testing to test the integration of using the STAC API within the application.

For each modular part within our system, an outline for each integration test will be provided as well as each Contract test that will be carried out. These outlines will encompass all of the individual components involved, along the expected output ensuring that the testing verifies all interactions within the application as expected.

3.1 Integration Testing

The integration tests that are listed below in this section will allow us to test different parts from different modules to ensure that the system is working together properly. The table below will show a designated input or action to be carried out within the GeoSTAC system and the expected output/outcome of the action. Each section listed is essential to test in order to provide a well functioning application to our users.

Input/Action	Expected Output/Outcome

Create and add a layer collection to the Astro Map.	The existing map will have new layers added to it.
Apply a Selected Area filter and apply it.	Have only footprints in the selected area be displayed on the map and in the results.
Filter footprints by a date range.	Have only footprints in the selected date range be displayed on the map and in the results.
Apply any search filter.	The Query Console will display the correct query string.
Clear the sort and filter menu.	The map will reset all footprints.

3.2 Contract Testing

The use of contract testing plays a key part in testing this application. With the use of an API to gather the STAC items of the scientific data to drive this application, it's critically important to verify that the data is successfully being retrieved. Below are outlines of interactions that will be tested between the GeoSTAC application and the STAC API. These tests or "interactions" will have a trigger event and an expected response or outcome. The purpose of implementing contract testing is not to test the STAC API itself or the GeoSTAC application, but to test the interactions and transmission of data between the two.

Event/Trigger	Expected response/outcome
Query API to filter footprints being returned.	Receive a smaller subset of footprints that correlate with the imputed filter.
Query the API for a GET/stac command	Receive the root STAC Catalog as a whole.
Filter footprints via bbox parameters (area of interest)	Have a list of footprints that only intersect with the selected area in any form.
Query the API for multiple filters at once.	Receive a subset of footprints that match up to numerous filter parameters.

4 Usability Testing

A large component of this web application is the usability on the frontend user interface. Therefore usability testing is essential to ensuring that the interface is easy to understand and maneuver. Since this web application will be open to the public and accessible by a wide range of user types, there will be user groups both within USGS and also general users outside USGS. Both groups will have a generalized set of instructions. These instructions will be provided in a Google form. Each step will be requested to be ranked 1 through 5 based on difficulty of completing each task. Each task will also have an option to apply comments or suggestions and specify any bugs potentially encountered.

There will be two sections of the usability test. First will be steps for interacting with the web map including using existing CartoCosmos functionality. The second section is steps for interacting with the footprint filter section and interaction with the footprint results. These two sections will help the team separate potential existing functionality changes and changes with new updates that are specific to the footprints. The following subsections will go into further details of what steps will be provided to the users for testing.

4.1 Testing Existing CartoCosmos Usability

Since this web application utilizes existing functionality from CartoCosmos, testing these functionalities will ensure that the usability is maintained. Some of the previous functionality was removed to provide simplicity to the modified application. This was largely due to some functionality no longer being needed for the STAC API interaction. So this testing is also useful for some users that have had previous experience with the older version of CartoCosmos. The following table provides each task that will be presented for user testing along with the expected outcomes:

User Tasks	**Expected Outcome**
Have the user access the deployed web app at: https://geostac.github.io/CartoCosmos-with-STAC/	Expect no issues with accessing the site. If there were issues with accessing this site, the user testing would end with this task.
Have the user select Mars as the Target Body	The expected outcome is that the user will

(This is the default target).	figure out that when initially accessing the github.io site, that Mars is the default body. This test will see if there are any confusions with that body that is currently selected.
Have the user change the base map of Mars to THEMIS IR Day.	The user will be able to identify where the layers button exists on the map and be able to change the layer in the toggle section.
Have the user recenter the map by using the Leaflet utility buttons.	The expected result is that the user will be able to find the button with the dot and arrows pointing at the dot and determine this is the centering button.
Have the user hover their mouse over the map and see coordinates.	The expected result is the user can see latitude and longitude boxes change according to the mouse movements.
Have the user zoom in and out of the map.	The expected result is that the user can either zoom in with a mouse scroll or use the plus and minus buttons within the Leaflet utility buttons.

4.2 Testing STAC Footprint Filtering Usability

This section of the usability testing focuses on the new features implemented for interacting with the STAC API filtering and the rendered footprints. The following table outlines each task presented to the user alongside the expected outcome for each task:

User Tasks	**Expected Outcome**
Have the user find the section of the Mars map that is displaying footprints. Have the user click a footprint to display the image. Have the user click the footprint again to remove the image.	The expected result is the user can view images within the webmap.
Have the user click the bounding box selection tool and make a selection around the section of existing footprints. Then have the user click the “Selected Area” box in the filter panel and click “Apply”.	The user should then see new rendered results based on bounding box selection. This also should show a results panel of the associated footprint information from the STAC API.
Have the user type in specific dates and click “Apply”.	Have a specific number of footprint results appear based on the date entered.

Have the user open the Query Console and click “Copy Code”	The user should be able to paste the link into a browser or terminal.
Have the user toggle on and off the mission specific footprint collection layer.	The user will be able to see the footprint display disappear when toggling this button.

5 Conclusion

As the need for accurate planetary data and images increases for the scientific and academic community, it is more important than ever that there is software to easily access this data. The USGS Astrogeology Science Center in Flagstaff Arizona provides the international planetary science community with new knowledge of our solar system. Without this data, new planetary exploration missions would be nearly impossible.

Now that the team has completed all of the major requirements for this project, the next step is to implement the testing plan mentioned above. This includes adding unit tests within the existing testing framework structure provided by CartoCosmos. This includes utilizing Jest, React Testing Library, and Mocha libraries for these unit tests. The integration tests will test the interactions between modules and contract tests will be used for testing the interaction with the STAC API. Finally, usability testing will be implemented through Google forms, administered to both internal USGS users as well as outside users. Both groups will be given the same set of instructions with a ranking of difficulty between 1-5. These tests will both ensure that the code base established is maintained when future updates are made and also ensure that the usability is consistent with a wide variety of users.