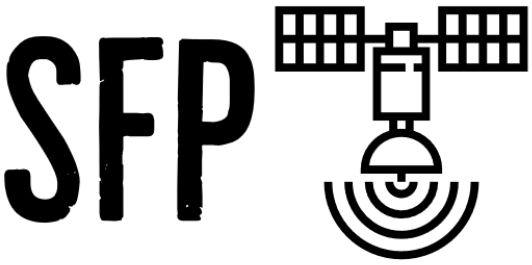
Team Satellite Fire Patrol

Software Testing Plan v1.1
January 26, 2024



Team Members:

Aaron Santiago, Ricardo Chairez and Zachary Hallemeyer

Clients:

Benjamin Wiebe, Dr. Camille Gaillard, and Dr. Christopher Doughty

Mentor:

Saisri Muttineni

TABLE OF CONTENTS

INTRODUCTION

The Hawaiian Islands hold immense ecological and cultural value but are currently threatened by increasing hot and dry conditions caused by climate change. This threat was evident in August 2023, when multiple wildfires across Maui resulted in at least 106 deaths, billions of dollars in damages, and ecological destruction. These fires begin in grasslands, which are vulnerable to high temperatures due to increasing drought and low moisture. Other Hawaiian ecosystems, such as coral reefs and tropical forest canopies, are also threatened by rising temperatures resulting from climate change.

Team Satellite Fire Patrol is partnering with clients Dr. Christopher Doughty, Dr. Camille Gaillard, and PhD student Benjamin Wiebe to create a GUI web application that uses real-time satellite thermal data to identify warning signs in the Hawaiian Islands and send alerts to the proper authorities. The app will automatically aggregate and process satellite thermal data from multiple sources, compare historical averages to highlight temperature anomalies, and present the data in a user-friendly interface for a range of resource management applications.

With these motivations in mind, extensive software testing is required. Software testing has the goal of ensuring our applications and components are working according to our expectations. As the web application increases in complexity and scale, it is incredibly important that testing be incorporated to validate the application as well as prevent bugs as more features are added or modified.

In light of this, this document will detail how the application and components are being tested, both manually and automatically. Some tools that are actively used in this project include Pytest for Python code and Chrome's DevTools to analyze our Vue application. The general plan for testing this web application and components will vary from component to component as they function and are made differently. Testing for our data handling component will consist of validating data from NASA Ecostress, ensuring dynamic error handling in data retrieval and processing. In regard to displaying the data, our vue web application will need to retrieve the data from our data storage component (NAU Monsoon) and display it in a timely manner. This will need to be accomplished with the use of a backend on our AWS server. In order to test this component, we will need to ensure that the backend and frontend will be able to dynamically handle errors in the case of network interruptions or miscommunications between Monsoon and AWS. The testing for the previously mentioned component will be elaborated and explored in this document.

UNIT TESTING

Unit testing is a vital component when building a software application. These tests ensure that all software components from the smallest functions to the biggest functions work as intended. More specifically, unit testing is the practice of confirming if a given function returns the expected result.

1.0 Monsoon

The software written to handle retrieving, processing, storing, and publishing the satellite data is written in Python and hosted in NAU's Monsoon. NAU Monsoon allows for fast computation provided with its vast components of hardware as well as a surplus of storage with the availability to be scaled up if needed. Python also allows for fast data manipulation with libraries such as NumPy and pandas. Because the code for Monsoon is written in Python, Pytest will be used for unit tests. This is because of the low base amount of code required to test each function. Pytest is also widely used in industry.

The code for retrieving data from external sources has the unfortunate nature of preventing 100% test coverage, as there may be unforeseen changes in API's. Therefore, testing for API's must rely on checking for correct status codes and confirming that data is being received. Furthermore, the code must be able to handle errors brought on through network interruptions and continue with computing without exiting execution. This means that the code retrieving data must be dynamic in nature to adjust to unforeseen changes.

In terms of data processing, a large part of validating incoming satellite images is ensuring that the data is from a coordinate on the Hawaii islands. This is accomplished with global_land_mask, which ensures that a given coordinate is a land coordinate rather than an ocean coordinate. Furthermore, the code validates coordinates by ensuring that they are within a bounding box containing Hawaii. Along with this, testing is performed with Pytest to ensure that a given satellite h5 file is converted to a geojson file, with land surface temperature being successfully converted to Fahrenheit or Celsius.

Cloud masking, removing clouds from satellite images, is also an important part of validating the temperature data that the site displays. In order to accomplish this, a cloud mask is also received from NASA ecostress and used to remove clouds.
In regard to data storage, the system is dynamic, so that the place in which data is stored can be easily changed and modified by simply changing data destination paths.

Finally, the processing data scripts run automatically on Monsoon on a time interval (base: 24 hours). Testing if the scripts are correctly scheduled is quite simple, as the scripts will be marked as a queued job.

1.1 Data Retrieval

- Tests for correct status codes from NASA Ecostress to confirm correct operations
- Ensure dynamic handling of errors for network interruptions
- Implement tests to confirm API response form

1.2 Data Processing

- Validate satellite image data with the use of global_land_mask as well as a limited bounding box of Hawaii
- Tests to confirm the temperature conversion from scaled Kelvin to Fahrenheit and Celsius
- Validate satellite temperature data with the use of cloud masking

1.1 Data Storage

- Validate that data retrieval and processing are correctly scheduled with the use of NAU Monsoon's job scheduler
- Test data storage is dynamic by changing data destination and processed data destination paths

2.0 AWS

Luckily for AWS there is only one section that needs to be tested. Inside the thermalwatch->scraper folder there is a scraper.js file which contains a web scraper that pulls data from the published monsoon website. To test the file, you simply run the file, and it will let you know directly from the terminal what is occurring and which files are being grabbed from the website.

For proper unit testing, we used the framework JEST, which is a JavaScript testing framework. This framework tests inputs and expected outputs, which came back looking optimal for the purposes of this project. JEST uses selenium and is widely used in large projects to test for user input which does not necessarily apply here.

INTEGRATION TESTING

Integration testing is to ensure that the different components of a software application work effectively together. For instance, in this project there is an AWS server that hosts a Vue application as well as a backend for said Vue application, and there is also the NAU Monsoon component that handles data processing. These components need to be connected together, and they need to be effectively tested to ensure that the connection can withstand error without crashing either component.

1.0 Monsoon and AWS

The data processing scripts publish the processed data to a constant URL. Therefore, the backend of AWS is able to retrieve the files at that constant URL. More specifically, a directory of available files is constantly updated by the scripts on Monsoon, which allows the backend to receive a dynamic and accurate list of all the data available for use. Therefore, the AWS backend serves as a bridge for the Vue application to use the data on Monsoon.

In order to ensure that errors are properly handled in the event of a requested file not existing or another unseen event. The backend will raise an error and continue with its execution. Similarly, due to the nature of Vue and JavaScript, the Vue application will not crash when a file is requested and not received. Rather, it will revert to a stable state.

2.0 AWS and Vue

Integrating Vue with AWS is an easy task with the built-in tools Vue has. Vue is node-based, so it runs using simple commands such as “npm run build” to build the application, which will not compile if there are any issues present. Then, once we resolve all the issues, we can run the command “npm run serve,” which serves the prebuilt application to the specified URL.

Using these methods, we will constantly test the application while adding features to it and ensure it works as expected. AWS is running Ubuntu, so everything is easy to navigate and test, which is why we will be constantly testing as we implement new features.

With the Vue framework, it also allows us to build the application to different specifications. We are currently building it normally, but once we have a finished product, we can build it for production, which is faster because Vue automatically optimizes our code and processes to run in a commercial environment.

USABILITY TESTING

Usability testing is focused on user interaction with our product. Our goals for this section are to ensure the user has the best experience possible using our product and to remove any bugs or unexpected behaviors. We will be mainly focusing on the website since the users will only have access to it, and we want to make it as easy to understand as possible. For our usability testing, clients and other students outside of our degree program will be testing out the website to provide helpful feedback on its usability. The clients will provide the most useful feedback since this product is being built for their needs. The students outside of our program will provide feedback for the

general public, which is also something that needs to be included because the clients want to release this project to the public.

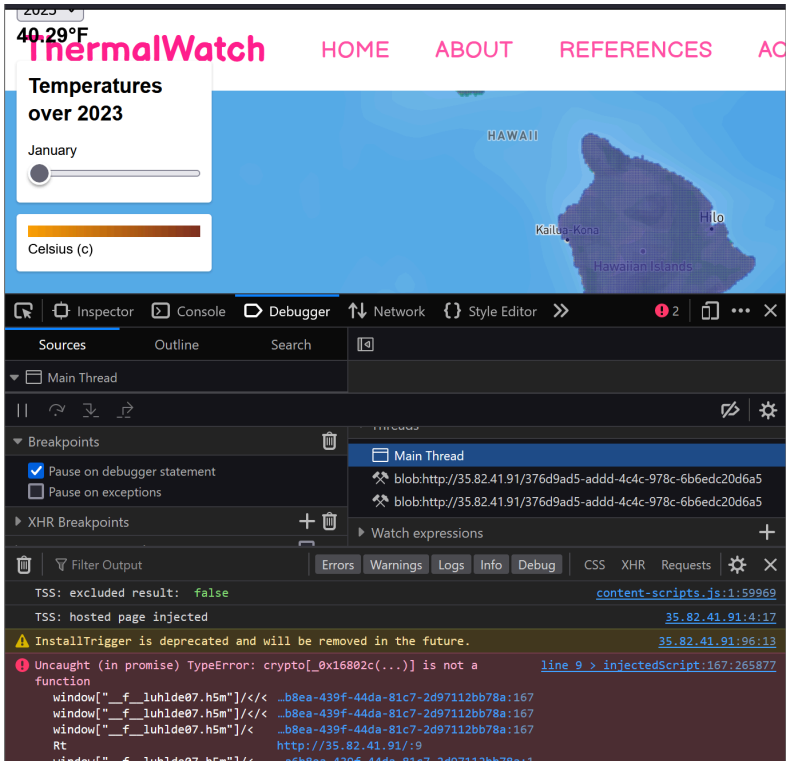
1.0 Monsoon

Monsoon itself will not be interacted with by the user, so we did not do any usability tests for it. Luckily our clients are well versed with monsoon and will feel comfortable using it for this project's purpose. All documentation and help related to monsoon as itself is not managed by us, so it is subject to change without notice.

2.0 AWS

Our website is hosted on AWS at thermalwatch.org. The website is fairly user-friendly, with more updates being applied. To check for user experience testing, we try to simulate what a normal user would do and reduce errors during development by using the inspect tool directly on the website. We will continue to present the website to the clients who can then recommend changes based on usability. Below is an example of what the inspect tool appears as with an error.

Figure 1: Inspect Element Tool



CONCLUSION

Overall, we are testing every part of this project individually and together to ensure all parts work as expected. Our project consists of two parts which are Monsoon and AWS, which we will continue unit testing as our project progresses to validate code behavior in both environments. Integration between the two environments is crucial for our project to work and is tested using fake files and real files to simulate a real working environment. Last but not least, our user experience is the most important part of this project because our final product is a website, which will also be the only part that is interacted with. We will test the user functionality ourselves by testing our features as soon as they are available and using the inspect tool on the website consistently.