Software Design

Version: 2.0

Team: Orion
Sponsor: USGS
2/15/19

Mentor: Isaac Shaffer
Client: Dr. Laszlo Kestay
Sponsor: USGS

Team Lead: Brandon Kindrick
Version Controller: Chadd Frasier
Code Designer: Yuxuan Zhu

Table of Contents

Introduction

1.1 Project Introduction

Planetary science is laying the foundation for humanity to become a space-faring species by discovering more about the geography of our neighboring planets. In recent decades, The National Aeronautics and Space Administration (NASA) has achieved more advancement in the field of planetary science than anyone could have predicted. What we can't ignore is the fact that the United States Geological Survey (USGS) has also played a vital role in this development. NASA works closely with the USGS to meet all of their data processing and research needs. The USGS has spent more than 30 years developing the Integrated System for Imagers and Spectrometers (ISIS) to decode and examine these massive binary files from NASA's imagers. ISIS decodes these files and retrieves the images, and metadata that goes along with them. Using the metadata, ISIS can be used for geospatial research. These precise topographic maps produced by ISIS are used every time researchers look for new potentially landing sites on the Moon, Mars and other distant planetary bodies. The USGS's contribution to space exploration is the accurate geospatial maps it produces, and without them you are lost in space.

1.2 Problem Introduction

We are team Orion and we have been given the opportunity to work with the USGS on the Planetary Image Caption Writing Project. Our client, Dr. Laszlo Kestay is a Research Geologist at the USGS. One of his regular tasks is producing publications on research he and his colleagues have conducted. This is where the problem resides. Dr. Kestay's current workflow for producing publication images is currently as follows:

- Load an image into ISIS
- Extract the Metadata
- Calculate by hand where he needs to crop image using ISIS qview
- Calculate how large his scale bar should be and draw it in a third party program, like Photoshop
- Repeat process if he needs to re-crop the image or needs to add additional graphics for metadata

This process is tedious and can take anywhere from 2-3 hours, making him reluctant to make images for his publications. This is where team Orion comes in. Dr. Kestay has tasked us with helping him streamline his publication image making process.

1.3 Solution Overview

Based on the requirements for this project, we have decided to implement a web app for our solution. By using a web app, our product will be usable on almost any OS platform. Using a web app will also give us access to all of the powerful web libraries that exist; like Openlayers, Node JS and Web Frameworks like Flask. Using a web app for our solution, means we will need to include ISIS with our app installation. ISIS is currently only available on Unix systems (Linux, OSX, etc.). To avoid alienating Windows users, we decided to use Docker as our installation method. Docker is a virtual "container" that acts like a virtual machine. With Docker, we plan to install linux (Ubuntu), and ISIS as well as our web app. This allows us to effectively install and run ISIS on Non Windows systems. To host our web app, we need to host it on a local server. To accomplish this, we are going to use the Flask framework. Flask, offers a wide array of tutorials and seems to be more beginner friendly than some of its alternatives. It also has extensive documentation and all of our team members know python. Finally our web app has to have the following functionality:

- Upload .cub images
- Crop .cub images
- Extract metadata from .cub images
- Add graphics to image based on extracted Metadata
- Export image
- Export metadata in CSV format and also in a specialized template used in publications

To implement this functionality with our web app, we plan on using base Flask and python for setting up the foundation of the web app. For implementing functionality of cropping and exporting of an image, we plan on using the Openlayers library. This treats base images as a "map". Map images can be easily cropped and exported to a number of formats. Because Openlayers is used for maps, it can also take into account more complex map phenomenon, like shortening a scale bar when close to a pole.

In summary, our solution will include the following:

- A Docker container that includes:
    - Ubuntu installed
    - A Flask web server hosting our front facing web app
    - ISIS
- A front facing web app with all functionality outlined above

1.4 Solution Conclusion

Our outlined solution covers all of the requirements of the project. The first issue being installation of ISIS on systems other than Unix. This is solved by us using a Docker container for our project. This allows us to install linux and ISIS in a small contained environment. The next major requirement is the ability to easily interact with ISIS. We accomplish this by implementing a web app that can communicate with ISIS. Due to it being installed in the same location as ISIS, it will be able to call ISIS commands from terminal using Flask commands. Being able to interact with .cub images and add graphics to them will be accomplished with the Openlayers library. To make the web app we implement available to the user, we are going to host it on a local server from the Docker container. The user will then be able to open the app from their browser.

Implementation

2.1 Implementation Overview

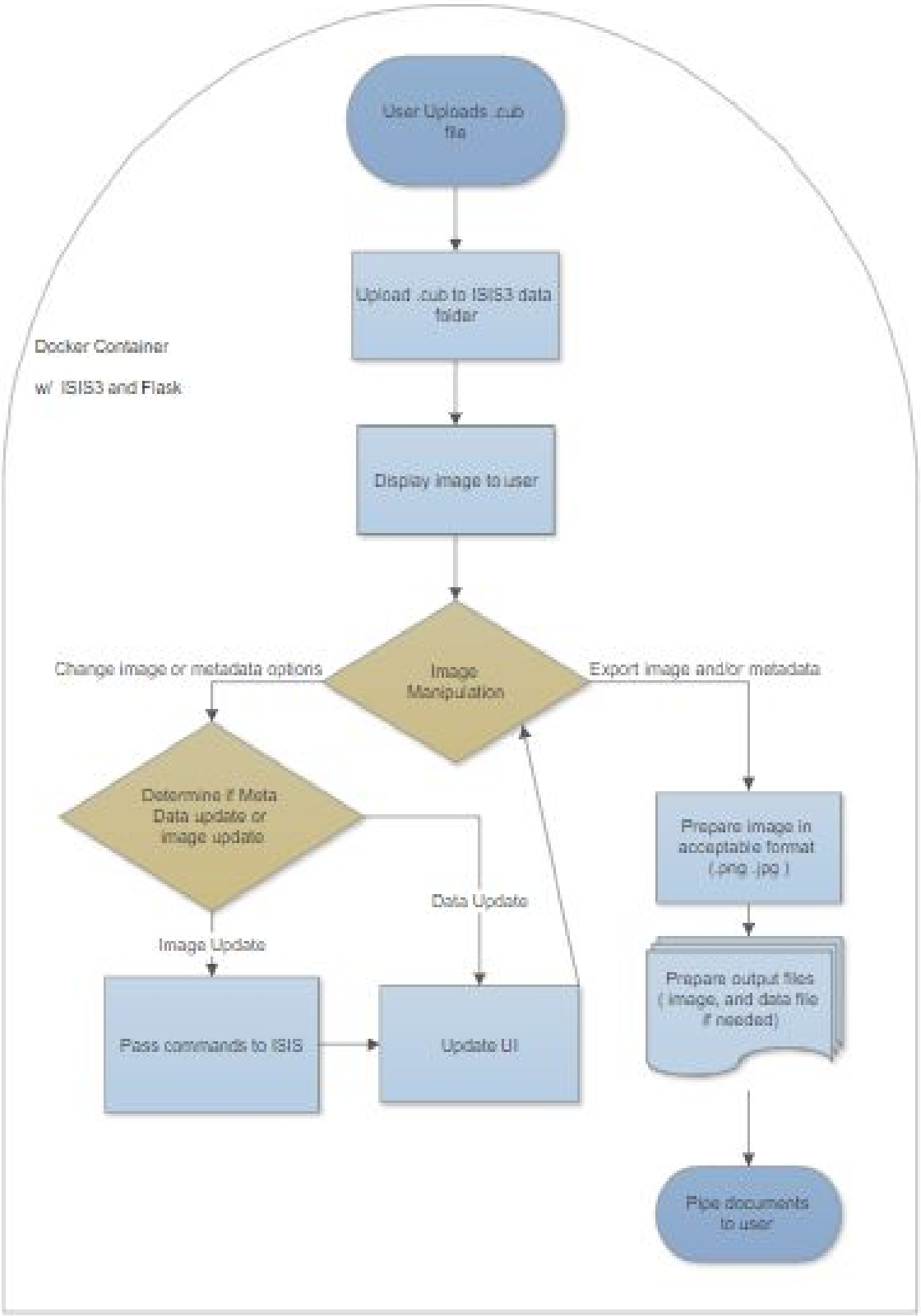
In accordance with the project's requirements, we have decided that a web app built with and hosted by Flask is the best solution because it is written in Python which all members of our group are familiar with and Flask provides a lighter framework in contrast to Django which is a very large framework. Our application will consist of two main portions; the Flask frontend, where users will give input and interact with buttons and images, and then the ISIS3 backend that will be performing all of the calculations on the files.

In order to make this application secure we will be developing it inside of a single Docker container. Docker is a lightweight, fast and extremely portable standardized environment that allows for development of web applications on a local environment. This allows for our tool to be secure in its own environment, with no overall access to the computer it is run on. This allows it to be run on any computer without security risks.

We decided that Flask would be the best option for building our user interface and front facing web app. Flask is a python framework that allows for building/ hosting web pages and apps on a local network. Compared to Flask's alternatives; like Django; is much smaller and easier to set up. Flask does not follow the Model View Controller architecture like Django does, which results in a much easier to install, lightweight application. Flask also is much easier to find resources and tutorials for due to the larger

user base. The primary reason for choosing Flask over the alternatives is availability of resources, the package size difference and because all of our group members know Python already.

Our application will be working off a client-server architecture where the clients can connect to a server that is running our Flask frontend. All the data from the user will be forwarded to the ISIS portion of the container and then returned to the client. We decided that ISIS will be run on the backend because doing this will allow us to use the ISIS command line functions from the Flask application. This will give us access to the suite of image editing functions provided by ISIS. ISIS can also natively extract metadata from .cub files. The approach outlined above is an overview of our web app implementation.


User Uploads .cub file
Upload .cub to ISIS3 data folder
Docker Container
w/ ISIS3 and Flask
Display image to user
Change image or metadata options
Image Manipulation
Export image and/or metadata
Determine if Meta Data update or image update
Prepare image in acceptable format (.png .jpg)
Data Update
Image Update
Prepare output files (image, and data file if needed)
Pass commands to ISIS
Update UI
Pipe documents to user

Architecture

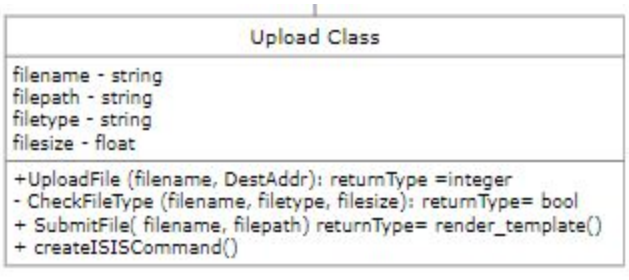
3.1 Architectural Overview

Starting from the beginning of our primary use case, the user will be prompted to upload a .cub file to set the whole process in motion. The system will be required to validate the file type of the upload before passing it to ISIS3. Once the file is validated for a proper file type, the file will be passed to ISIS3 to be deconstructed and stored for later access. In order to more easily manipulate the image from ISIS3 we will convert it to a more standardized image to allow for easier importation into Flask. After we have the image and metadata prepared we will use the Flask HTTP resources to post the new image and data to the web app. After the image is properly displayed the user will be able to crop, highlight, and add icons to the image. The user will have the freedom to export the image as soon as it has been properly uploaded. If the user chooses to manipulate the image we must decide what type of manipulation it is. If it is a data involved manipulation we can add icons or bars using the metadata. If it is a photo manipulation such as cropping we must pass the file and pixel locations to the cropping function on ISIS3 and return the newly cropped image and metadata using a template and passing it the new image file. If the user applies both image and data changes, we must manipulate the image before the data because the metadata will change after the image is cropped. After we display the newly updated web app we again will allow the user to further manipulate the image or the user can begin the exportation process. When the user starts exporting they must begin with a simple interface that configures the save file path and type. After the user selects an appropriate file type and path they can click a button to begin the export. We will then need to prepare the output files using functions that read the users output settings and prepares either the images or the metadata or both into proper file types.

Module and Interface Descriptions

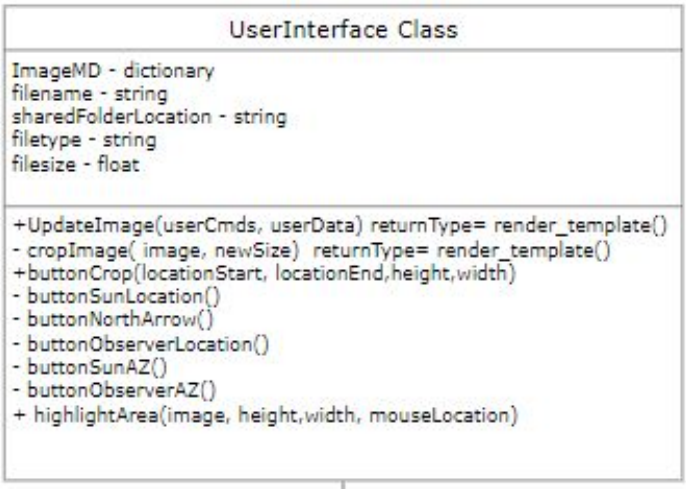
4.1 Upload Class

The system will begin with an Upload class that will be responsible for informing the user to upload an ISIS3 cube file and present a window interface for the file upload. We will achieve this by creating an UploadFile function that will utilize a helper function to verify the user's input. We will only be accepting cube files and if this function does not pass then the user will be



prompted to retry a new file. After the file is verified we will pass the file to a create Isis Command function that will format a sting that can be passed to an ISIS3 decoding function. After we create the command string we will submit it to the ISIS3 portion of the Docker container where a function can read the command in from the frontend and perform the necessary commands in the ISIS software. The main interface will receive the image and metadata from ISIS and store it on the web app for better access. The upload class will also be responsible for configuring the ISIS API and shared folder locations of the application. After a successful upload, we can parse through the cube metadata and retrieve the instrument that is currently being used, the commands that that instrument uses.

4.2 User Interface Class

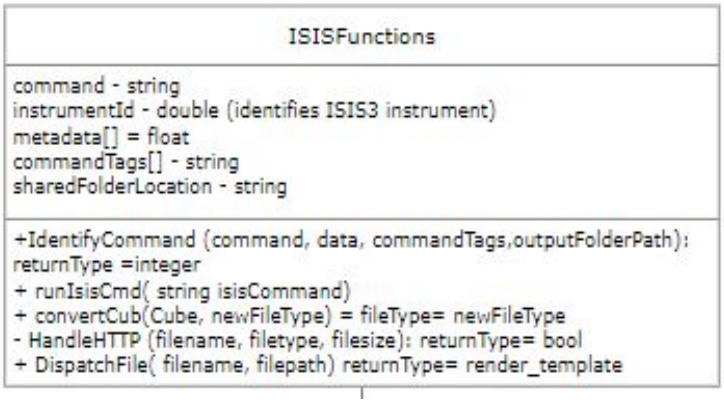


The user interface class will be responsible for all of the user interactions with ISIS. The user interface will consist of check boxes and buttons to add and remove icons from the image as well as a large photo view. Checkboxes will be used for selecting which metadata to display. Buttons will better for things like adding icons to the image because ISIS can help us calculate where it will need to appear. We will be using buttons to apply image changes and icons. The cropping feature for example will work in a similar manner to the cropping features in most word processors. A button with scissors on it that will change color or look in some way when clicked to denote that it is in cropping mode. When cropping we will track the mouse location and allow the user to create a rectangle outline on the image that will be the newly cropped image. Four coordinates will be sent to ISIS. To accomplish this, we will send the four coordinates and use the ISIS crop function to return the final cropped image.

We will need to validate every aspect of the the commands before sending it to ISIS3. We will associate buttons of the UI with batches of commands for ISIS3. For example the cropping feature would take the input and from the user in the form of a outlined rectangle we will save the size of the rectangle, the location of the top left corner of the box in the form of pixels away from the edge. We will then want to use the isis2std function to change the image format to a png that can be very easily displayed onto the web app. We will need these types of functions for adding every icon including the Sun azimuthal direction, a north arrow indicator and the observer location.

The user interface will also have a function that will add highlighted areas to the image to help the user bring attention to portions of the image.

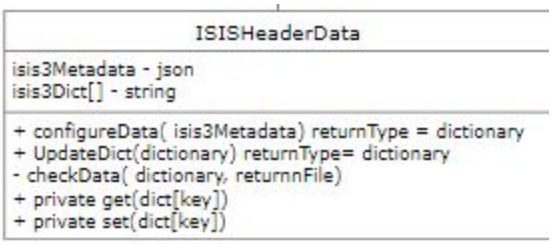
4.3 ISIS Functions Class



The ISIS control functions will be invoked by the buttons on the user interface and these functions will act as a restful api to perform the necessary actions on the cube file and return the new image. The api will need to verify each piece of the command that it receives and perform commands in the correct order when required to do so. After changing a cube, ISIS3 will need to create an image that Flask can display. Using an api to manage these processes will create a fast and easily maintainable interaction process between the web app and ISIS. Writing a simple api in Flask will create a very simple and fast HTTP handling system. The api will handle all the commands and return the image by running a conversion function on the image and icons to be included. We will need to run a function that sends the new data back to the client after the functions are run and the Flask GET and POST functionality will make this job simple.

4.4 ISIS Header Data Class

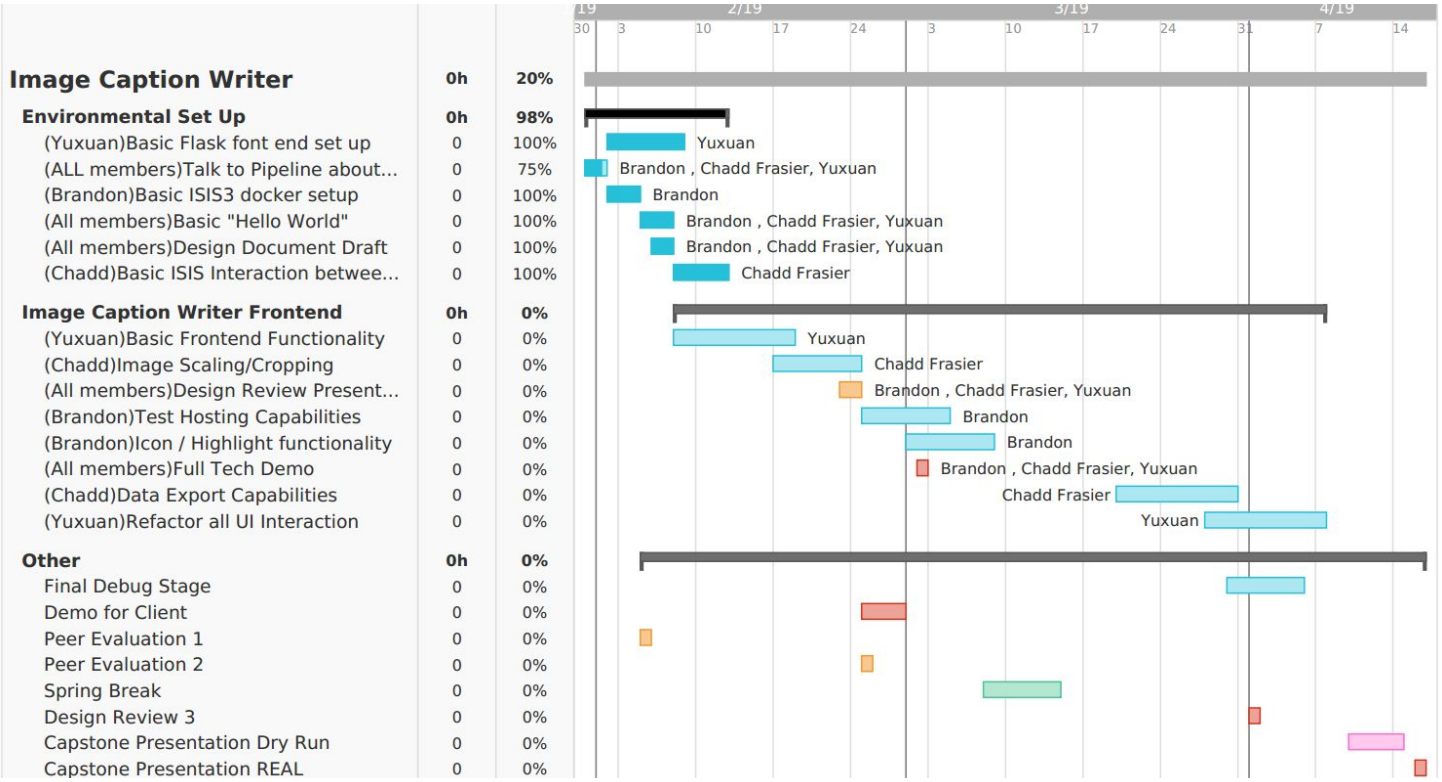
We will create a class that is devoted to storing and retrieving the ISIS header data that our application requires. It will be easier to efficiently store this data by creating a configuration function that will convert the ISIS header data from a json format to a python dictionary. We will use a dictionary to take advantage of the key-value format that python is capable of. Using a dictionary in this case will allow us to access any piece of metadata with the exact same variable name and a unique key. After the data is properly configured the ISIS header data class will be responsible for updating the dictionary if the values are changed, either by cropping or by rotating. This will require two functions, one for checking to see if the data new data is different than the current dictionary and a function to overwrite the dictionary if need be. In order to test and change the values we will need to write a generic get and set method that reads the input in an effective way so that we do not need more than one function for the data. The getter function will return the value of the key it is given, this means that



we will need to validate the output before passing it to other functions or comparing it against other values. The setter function will only be called in the update function to prevent from accidental overwrite.

Implementation

In this section, we are going to provide a design-centric implementation timeline for our project. We designed a gantt chart to reflect each required task related to the project. Using a gantt chart helps to plan, coordinate, and track specific tasks in our project. The following is our team's current gantt chart:



In the Gantt chart shown above, we have provided a detailed schedule of each module in our project. Each task is assigned to a specific team member who will lead the development of said task.

The first task is the Environmental setup. This is the basis of our mission and includes several modules. As we mentioned in “Implementation Overview,” we will use Flask for our front end, and ISIS3 on our backend to perform all of the calculations needed. Both the front end and back end are included in the Docker container, ensuring security of our app and availability on most OS platforms. The USGS has created a Docker container image on Github that provides everything needed to run our web application: code, runtime, system tools, system libraries and settings. Our first task is to setup our Docker environment and establish basic ISIS/Flask communication. We have installed the ISIS Docker image on our laptops and installed Flask inside the Docker container. The next step is to build a basic communication messaging system between Flask and ISIS3 using Flask to call ISIS commands at runtime.

Our second major task, as well as our full tech demo step, is “Image Caption Writer Frontend.” We will build basic frontend functionality which includes all the features our customer required. The main features will include cropping the image, adding standardized graphical symbols for specified geospatial information, retrieving additional geospatial information in a subsection, extracting the data in a readable format, and finally exporting the user’s image with all the added icons and data pieces. After implementing these functions, we scheduled time to test hosting capabilities for our web tool. Because we use a Docker container and a Flask server inside, we must make sure all these things work and will have a long life.

Finally, in the “other” section, includes two very important tasks. They are debugging our web application and showing it to our client. We will need to ensure that Dr. Kestay is comfortable with our installation process and with the functionality of our UI.

Conclusion

We are team Orion, and we are working with the United States Geological Survey (USGS) on the Planetary Image Caption Writing Project. Our client is Dr. Kestay, who is a Research Geologist who has worked at the USGS for many years. It is well known that images in space publications are at the heart of sharing information and gaining public interest. The problem our client is facing now is having no convenient way to extract the metadata contained in images other than by ISIS3 command line calls and a pencil. Dr. Kestay has to manually extract the metadata by using the binary editor on the ISIS3 file and then type into a document. If he wants to build annotations

on the image, he has to use Photoshop. The whole procedure takes a long time and is incredibly tedious.

To solve this problem, Our team plans to repackage ISIS and make it accessible on any platform via a web tool. It will include a user friendly “viewer tool” on the front end. Our goal is to make the app accessible to anyone whether they are an experienced programmer or someone who has never touched a programming language. We will use the Flask framework to develop our front-end that allows users to process an image, edit metadata, and finally, export the user’s image with all the added icons and metadata. For the backend, we will use ISIS3 because it already has all of the functions we want for our web tool. It will process requests sent by front-end and run corresponding commands then send the result back to the user side. We will develop this web application inside of a single Docker container. Docker allows our team to package up our web application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. As a result, our web application will run on any machine, regardless of OS installation.

Through this document, we have outlined the overview of our product. By going over our client’s requirements and his problems, we have found the problem. Our solution overview addressed each part of the problem and how our solution address it. We have also gone over the technical layout for our product. We addressed how we plan to implement our solution. Following the schedule reflected in our Gantt Chart, and by following the design of our project, we are confident we can provide the product that will satisfy our client’s requirements and better the production of publication ready images.