Josh Surette
February 19, 2018
EC500
API Assignment 2

<u>Description of Work</u>

Zhiwei's API:

I was assigned to work on Zhiwei's API at https://github.com/tzwk/EC500-C1. I created my own branch and started to begin work. However, I noticed that all of the functionality was allocated to one long Python script with a lot of duplicate code and used AWS, which I wasn't familiar with. I started to begin to try to rewrite the code to create an actual API out of it that a front end application could call, but found that with time constraints this wouldn't be practical. I did an extensive code review and left 17 issues which covered convention, refactoring, and covering edge cases that may arise, but due to the state of the code I did not write a front end application or unit tests since there were no actual functions and the file was just a standalone script and not an API.

My API:

Professor Alshayk gave me the okay to then go ahead and write unit tests and a front end application for the backend I wrote. The backend I wrote was written with Dropwizard, which is a Java framework used to write rest APIs. I started off here by creating my own branch at **https://github.com/jrs33/videoMaker/tree/unit-test-UI-implementation**.The backend was broken up into three major endpoints: the /twitter endpoint to get image url addresses, the / google endpoint to POST image addresses for annotation, and the /ffmpeg end point to both save images locally and create a local video. I wrote unit tests for each of these endpoint groups (3 in total) and covered main things in each group, including

1) Passing in empty JSON and invalid JSON objects to the POST requests to check when they would break, and also passing in invalid query and path parameters (ie: invalid tokens, invalid directories) to try and break the code as well
2) Checking for the runtime of the endpoints when called properly
3) Checking the memory usage of each of these endpoints

These tests were also automated given that the Maven build system automatically runs @Test annotated test scripts before it builds fat jars. I was only able to easily break one function, which was the getImagesFromTweets function, when passing in empty image JSON. However, other than this, many of the functions already took advantage of built-in exception handling from the API calls, so there weren't many other cases where the code was broken from strange inputs. I was also able to gather the following data from a trial testing run:

Functions	Twitter Image Retrieval	Google CV Tagging	Saving Images Locally	FFMPEG Making Video
Time (ms)	470	702	414	33
Memory(bytes)	20396896	11398968	4125480	5558824

We can see here, due to authentication and overhead caused by using the external API's, we had a lot of memory usage and latency compared to the local operations of video creation and image saving.

After this, I was able to create a front end UI using Dropwizard Views, and use the Freemarker Templating Language from Apache. I created a client endpoint here called / createVideoUserInterface and used an instantiated Jersey client to make calls to my API, which allowed me to get data retrieved. Here it was clear that the Twitter endpoint worked well and never broke the program. However, the Google endpoint returned a malformed JSON since it was in a list (ie: enclosed in [] rather than {}) which meant the response could not be processed by my client. Thus, the tagging here would not work which would need to be fixed. Along with this, the config.yml was not setup with the credential and path constants, so these needed to be hardcoded in the code itself, which should be changed. It was also clear that saving the image locally was a problem. The HTML5 tagging in the ftl language expects to play a video from a URL, so saving it locally was a problem. There is also the case that, when the video doesn't actually get created, or images aren't located in the path, the program doesn't crash but it also gives no warnings; it just progresses and gives no feedback to the user. I was able to find a lot of these non-obvious errors I did not catch in my simple unit tests as I created the front end and created issues for them as well to allow for a better front end application to be developed.

Overall, the API functioned relatively well when isolating back end tests and unit tests for output, but some of the output was unusable by clients, and some of the misbehaviors need to be flagged better to provide a clearer experience for the user.