CS Capstone Final Report

June 9, 2017

Aerosol Analyzer Mobile Web Application

Prepared for

NASA JPL

Dr. Kim Whitehall _____ _____
Signature *Date*

Prepared by

Group 22
Aerolyzer

E. Reilly Collins _____ _____
Signature *Date*

Sophia Liu _____ _____
Signature *Date*

Jesse Hanson _____ _____
Signature *Date*

Abstract

Over the past few months, the Aerolyzer team has worked with our client to make progress towards our finished software product. The purpose of this document is to illustrate that progress and provide a retrospective of our work done so far. An introduction of our project, a discussion of changes to our requirements, design, and tech review documents, and introspective of our learning throughout the project are included. Additionally, this document contains our weekly blog posts, final poster, and project documentation. After demonstrating our aerosol-analyzing mobile application at Expo, this document serves as a thorough overview of our Aerolyzer project.

In summary, we have completed several documents that aided us in figuring out how our project will come together and made changes to these along the way. Furthermore, we worked on several tasks assigned by our client and added these to our code repository comprising of our final product. Lastly, we have continued making progress this term towards a better understanding of software design, the development process, and programming collaboration.

CONTENTS

List of Tables

List of Figures

I Introduction

Aerolyzer is a project that was requested by client Dr. Kim Whitehall in collaboration with Dr. Lewis McGibbney. She requested this project to develop an application that supports the ability to understand content from images, and encourage human curiosity in nature. The development of this application is important because provides citizen scientists with the ability to understand the aerosol content of their image, as well as provides scientists with a large database of aerosol data throughout the globe. The members of our team consist of Oregon State University students Reilly Collins, Sophia Liu, and Jesse Hanson, as well as clients Dr. Kim Whitehall and Dr. Lewis McGibbney. As for the students' individual roles, Reilly and Sophia focused mainly on frontend and database development, while Jesse focused on developing the backend image processing. The role of clients Dr. Kim Whitehall and Dr. Lewis McGibbney was mainly to supervise and facilitate our progress, although Dr. Kim Whitehall also contributed to the development of our project.

II Requirements

II.1 Original Requirements Document

Software Requirements Specification

for

Aerolyzer

Version 1.0 approved

Prepared by E. Reilly Collins, Sophia Liu, and Jesse Hanson

Group 22

CS 461 Fall 2016

November 7, 2016

Abstract: This document contains the software requirements specification for Aerolyzer. Aerolyzer is a mobile Web application capable of processing visible images and inferring atmospheric phenomena to provide the general public with near-real time monitoring of aerosol conditions. The goal of this document is to clearly specify the requirements for our mobile app to be developed.

CONTENTS

I. Introduction

A. Purpose

The purpose of this project, Aerolyzer (version 1.0), is to develop a mobile web application that supports the ability to understand content from user-uploaded digital outdoor images. Our goal is to allow the general public to better monitor current aerosol content.

B. Scope

This software will be a mobile web application for use by the general public. This application will be designed to uniquely utilize color distribution within an uploaded mobile image to identify features necessary for analyzing aerosol content. The end product will be a full mobile web app that employs an open source algorithm translating how the human brain interprets images using RGB values. Citizens in the general public will be able to quantify color in images and use the resulting data to determine current aerosol content.

More specifically, users will be able to upload an image of an outdoor scene taken on a mobile phone. The application will then give output detailing the current aerosol content of the atmosphere based on the uploaded image. This software will provide average citizens with near-real time monitoring of atmospheric conditions.

C. Definitions, acronyms, and abbreviations

Aerosol: Minute particles suspended in the atmosphere. When these particles are sufficiently large, we notice their presence as they scatter and absorb sunlight. Their scattering of sunlight can reduce visibility (haze) and redden sunrises and sunsets. [1]

EXIF/exif: Exchangeable image file format specifying the specs of digital/smartphone cameras.

MISR: Multi-angle Imaging SpectroRadiometer satellite. [3]

More definitions, acronyms, and abbreviations will be added as needed to this requirements specification document.

D. References

[1] Bob Allen. (1996) Atmospheric Aerosols: What Are They, and Why Are They So Important? [Online]. Available: http://www.nasa.gov/centers/langley/news/factsheets/Aerosols.html

[2] WeatherAPI: Introductions. [Online]. Available: https://www.wunderground.com/weather/api/d/docs

[3] MISR: Home Page. [Online]. Available: http://www-misr.jpl.nasa.gov/

E. Overview

The next section, Overall Description, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next section. The third section, Specific Requirements, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product.

Both sections of the document describe the same software product in its entirety, but are intended for different audiences and thus use different language.

II. Overall Description

This section will give an overview of the whole application. The app will be explained in its context to show how it interacts within a larger system, as well as introduce basic functionality. It will also describe the type of stakeholders who will use the app and what functionality is available for them. At last, the constraints and assumptions for the system will be presented.

A. Product Perspective

Aerolyzer will be an independent and self-contained mobile web application. It is not designed to be a component of a larger system. Mobile images and associated metadata used for the color-analyzing algorithm will be from various online sources, such as Weather Underground.

As a note, Aerolyzer will be similar to other citizen science applications, though not part of those applications, by allowing user images to be used in improving the machine learning of the algorithm. More information about the citizen science component of the system will be added as needed to this requirements specification document.

B. Product Functions

Within the mobile application, users will be able to upload an image of an outdoor scene with the sky in view from their phones camera roll. An algorithm will use the image's colors and associated metadata, combined with other meteorological data, to output current atmospheric conditions and aerosol content. Output from the color-analyzing algorithm will be displayed on the screen for the user.

C. User Characteristics

1) Users will be smartphone owners.
2) Users will need technical knowledge about the basic functionality of their smartphones to take photos.
3) Users can access the application on their desktop computer or smartphone, but will need to know how to use a web browser and navigate to the Aerolyzer app for both interfaces.
4) Users will need to have access to mobile photos whether they are on the smartphone or desktop version of the web application.
5) Users educational levels can be anywhere across the board; they do not need expertise in aerosols in order to understand output.

D. Constraints

1) Although the system is a web application, uploaded images must be taken on phones in order for the algorithm to give correct output. Images from mobile phones include associated metadata that will be needed for the aerosol analyzing algorithm, and this metadata is not present on non-mobile cameras.
2) Images uploaded to Aerolyzer cannot be cleaned up before being uploaded, as this could distort the color. This means no filters or other image editing tools can be used on the images.
3) Only direct landscape images with the sky in view can be uploaded to Aerolyzer. For example, images taken from inside a window or a picture of grass will not give accurate output about current aerosol conditions.

More information about constraints of the system will be added as needed to this requirements specification document.

E. Assumptions and Dependencies

1) We assume users are willing to give permission for having their images accessed and processed.
2) We assume that users have access to at least one mobile image in their photo library to be uploaded.
3) We assume that users are connected to the Internet.
4) We assume users have access to a web browser.

III. Specific Requirements

This section will give an overview of specific requirements for the application. Links to external systems will be explained to show what other resources the application will use. The app's functions in terms of individual use cases will be described in detail.

A. External Interfaces

One link to an external system will be accessing weather information from the Weather Underground database using their Weather API. We will need this to gather astronomy data, almanac information, and current conditions (at a minimum). The output from this API will be used as input by the algorithm that determines current aerosol content. [2]
Another link to an external database will be pulling outdoor images from online sources to be used in the color-analyzing algorithms machine learning. The source of these images will be from reputable repositories of images that include associated mobile camera metadata, such as Weather Underground. The purpose of these images will be to use them as input in order to improve the algorithms machine learning component.
A final link to an external interface will be the access to stored images and associated output from the color-analyzing algorithm. More information about image and data storage will be added as needed to this requirements specification document.

B. Functional Requirements

1) Web Application Interface:

Trigger: Users access the Aerolyzer mobile web application.

Precondition: Users are on a mobile device with access to a web browser and the Internet.

Basic Path:

1) The user opens a Web browser on his or her mobile device.
2) The user navigates to the Aerolyzer URL using the web browser.
3) The system shall load the start page on the users web browser.

Alternative Paths: Users can access the system by clicking a link that opens up their mobile web browser.

Postcondition: The user has access to the Aerolyzer mobile web application.

Exception Paths: If the user does not have Internet access, the use case is abandoned.
If the user is trying to access the system from a desktop device, the system alerts the user that only mobile images are allowed and the use case is abandoned.

2) Upload Photo:

Trigger: User selects button to upload a photo to be analyzed.

Precondition: Users are on a mobile device with at least one image in its photo library.

The web browser used to access the system has access to the users mobile photos.

Basic Path:

1) The user selects the Upload a Photo option from the systems start page.
2) The mobile devices photo library is displayed with the option to select one photo to be uploaded.
3) The user selects one photo to be uploaded and then selects the Use Photo button.

Alternative Paths: None.

Postcondition: The user is taken to a loading page while the photo is being analyzed.

Exception Paths: In step 3., the user may abandon the upload at any time by selecting the Cancel option.

If the photo being uploaded does not meet constraints identified in section 2.4, the image cannot be used as input for the color-analyzing algorithm. The user will be notified and the use case is abandoned.

3) Display of Aerosol Content:

Trigger: Photo has been analyzed by the algorithm and the resulting aerosol content has been determined.

Precondition: A photo has been successfully uploaded as per use case 2.

Basic Path:

1) A loading screen is displayed and the user waits for the photo to be analyzed.
2) The photo is used as input for the color-analyzing algorithm.
3) The algorithm outputs data for current atmospheric conditions.

Alternative Paths: None.

Postcondition: The user is taken to a page where the resulting aerosol content of the photo uploaded is displayed.

Exception Paths: In step 1., the user may abandon the analysis at any time by selecting the Cancel option.

4) Extract EXIF Information From Data:

Trigger: A script that extracts EXIF data from an uploaded image is run.

Precondition: A photo has been successfully uploaded as per use case 2.

Basic Path:

1) The Aerolyzer developer receives images as input for the script.
2) The script outputs a JSON file containing the filename as a string, the category of/tags for the photo (e.g. clouds, fall colors, sunrise, etc.) as a list, and the EXIF data associated with the image as a dictionary.
3) The JSON file retrieved in step 2 is used as input for the core color-analyzing algorithm and is stored in the central storing location.

Alternative Paths: Images and associated metadata from any source can be manually added to the central storing location.

Postcondition: Aerolyzer's image repository includes retrieved images for use in improving the color-analyzing algorithm.

Exception Paths: None.

5) Using Weatherunderground API:

Trigger: Picture is uploaded and ready to be analyzed.

Precondition: A photo has been successfully uploaded as per use case 2.

Basic Path:

1) Meteorological and astronomical data is retrieved from the Weather Underground API.
2) The information derived from the API is given as input to the color-analyzing algorithm.
3) The uploaded image is analyzed via the algorithm.

Alternative Paths: None.

Postcondition: The color-analyzing algorithm runs using Weather Underground API data and the user is taken to a page where the resulting weather information based on the photo uploaded is displayed.

Exception Paths: If data is unable to be retrieved from the Weather Underground API, this data will not be used as input for the color-analyzing algorithm and may affect the accuracy of results.

If data is unable to be retrieved from the Weather Underground API, this data will not be displayed to the user when viewing aerosol content results.

6) Get Data From Satellite Source:

Trigger: Picture is uploaded and ready to be analyzed.

Precondition: A photo has been successfully uploaded as per use case 2.

Basic Path:

1) Geo-related data is retrieved from the satellite source (such as MISR) based on the iamge's location.
2) The information derived from the satellite source is given as input to the color-analyzing algorithm.
3) The uploaded image is analyzed via the algorithm.

Alternative Paths: None.

Postcondition: The color-analyzing algorithm runs using satellite data and the user is taken to a page where the resulting satellite information based on the photo uploaded is displayed.

Exception Paths: If data is unable to be retrieved from the satellite, this data will not be used as input for the color-analyzing algorithm and may affect the accuracy of results.

If data is unable to be retrieved from the satellite, this data will not be displayed to the user when viewing aerosol content results.

7) Running the Core Algorithm:

Trigger: Picture is uploaded and ready to be analyzed.

Precondition: A photo has been successfully uploaded as per use case 2.

EXIF, meteorological/astronomical, and geo-related data has been successfully extracted as per use cases 4, 5, and 6, respectively. **Basic Path:**

1) The application uses EXIF, meteorological/astronomical, and geo-related data as input for the core color-analyzing algorithm.
2) The core algorithm runs.

Alternative Paths: None.

Postcondition: The output from the algorithm is used to display aerosol content as per use case 3 and is stored in the central storing location.

Exception Paths: If the needed meteorological data is not available, the algorithm will still run and give output that does not include the meteorological data.

C. Performance Requirements

The web application will have both mobile-scaled and desktop-scaled interfaces.

The system will be able to analyze at most 500 images per day total and at most 10 images per minute total (meaning for all combined users, not each individual user).

More performance requirements will be added to this requirements specification document after further research on the color-analyzing algorithm capabilities is complete.

SIGNATURES

Kim Whitehall

Client, NASA JPL

Signature: _____

Date: November 7, 2016

E. Reilly Collins

Student, Oregon State

Signature: _____

Date: November 7, 2016

Sophia Liu

Student, Oregon State

Signature: _____

Date: November 7, 2016

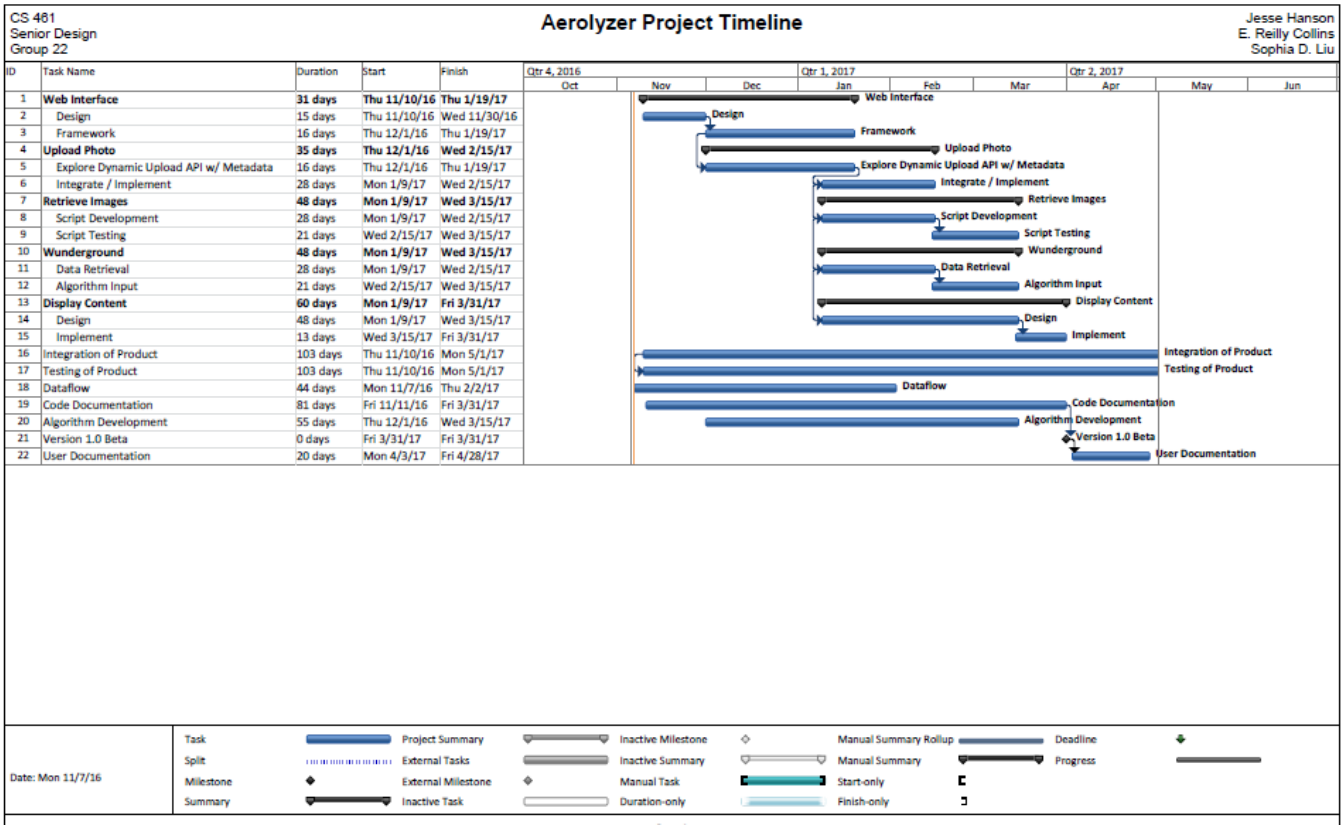
Jesse Hanson

Student, Oregon State

Signature: _____

Date: November 7, 2016

II.2 Original Gantt Chart



ID	Task Name	Duration	Start	Finish
1	**Web Interface**	**31 days**	**Thu 11/10/16**	**Thu 1/19/17**
2	Design	15 days	Thu 11/10/16	Wed 11/30/16
3	Framework	16 days	Thu 12/1/16	Thu 1/19/17
4	**Upload Photo**	**35 days**	**Thu 12/1/16**	**Wed 2/15/17**
5	Explore Dynamic Upload API w/ Metadata	16 days	Thu 12/1/16	Thu 1/19/17
6	Integrate / Implement	28 days	Mon 1/9/17	Wed 2/15/17
7	**Retrieve Images**	**48 days**	**Mon 1/9/17**	**Wed 3/15/17**
8	Script Development	28 days	Mon 1/9/17	Wed 2/15/17
9	Script Testing	21 days	Wed 2/15/17	Wed 3/15/17
10	**Wunderground**	**48 days**	**Mon 1/9/17**	**Wed 3/15/17**
11	Data Retrieval	28 days	Mon 1/9/17	Wed 2/15/17
12	Algorithm Input	21 days	Wed 2/15/17	Wed 3/15/17
13	**Display Content**	**60 days**	**Mon 1/9/17**	**Fri 3/31/17**
14	Design	48 days	Mon 1/9/17	Wed 3/15/17
15	Implement	13 days	Wed 3/15/17	Fri 3/31/17
16	Integration of Product	103 days	Thu 11/10/16	Mon 5/1/17
17	Testing of Product	103 days	Thu 11/10/16	Mon 5/1/17
18	Dataflow	44 days	Mon 11/7/16	Thu 2/2/17
19	Code Documentation	81 days	Fri 11/11/16	Fri 3/31/17
20	Algorithm Development	55 days	Thu 12/1/16	Wed 3/15/17
21	Version 1.0 Beta	0 days	Fri 3/31/17	Fri 3/31/17
22	User Documentation	20 days	Mon 4/3/17	Fri 4/28/17

Fig. 1. Original Gantt chart for requirements

II.3 Client Requirements Document Changes

The basic functionality of our product outlined in our original requirements document remains unchanged. However, one major aspect of our project that differs from the description in this document involved the actual core color-analyzing aerosol algorithm. Our client relayed to us that she overestimated our time commitment to the project, as well as the freedoms for her to guide the project as the client. Instead of working on both computer science and atmospheric science aspects of our development, we chose to focus only on the computer science component. This change means our final product only met requirements 1-5 (rather than all 7), and ”Display of Aerosol Content” requirement 3 does not actually display output from the core algorithm yet. The atmospheric science component will be completed by a capstone team next year.

Our ”Extract EXIF Information From Data” requirement 4 also changed, essentially being split up into two different EXIF-related requirements. A script was written that scrapes the category and tags from online images as a list, then outputs this data into a JSON file to be stored on Github and used to improve the color-analyzing algorithm. Our webapp itself also has a script that retrieves EXIF data from uploaded photos and returns required EXIF data as a dictionary to be used in the core algorithm; all photos and EXIF data are stored in our image repository as described in the requirement.

Other (mostly minor) changes to requirements are listed as comments in table 1.

TABLE 1
Functional requirement changes

Num	Requirement name	What happened	Comments
1	Web Application Interface	Removed mobile-only restriction; instead, users can access website via desktop and are simply notified if uploaded image does not meet mobile-only restriction. This change was made to reach a wider range of users after discussion with our client.	Functionally unchanged.
2	Upload Photo	Minor text difference: ”Use Photo” button changed to ”Submit” button as a simple UI decision.	Functionally unchanged.
3	Display of Aerosol Content	Actual aerosol content is not displayed due to core algorithm not being implemented until phase 2; however, the page is displayed as described and there is a blank box where aerosol content will eventually be shown. Minor text difference: ”Cancel” button changed to ”Upload” button cancelling current upload as simple UI decision.	Functionally unchanged.
4	Extract EXIF Information From Data	Split into two EXIF-related requirements. This change was made as our client needed both test data to improve the core algorithm (e.g. verify the algorithm could correctly detect sunsets from an image with a ”sunset” category) and actual EXIF data from an uploaded image to get that image’s aerosol content.	Details explained wholly in above subsection introduction.
5	Using Weather Underground API		Functionally unchanged.
6	Get Data From Satellite Source	Removed in phase 1 (development phase). Will be implemented in phase 2 (atmospheric science phase).	Details explained wholly in above subsection introduction.
7	Running the Core Algorithm	Removed in phase 1 (development phase). Will be implemented in phase 2 (atmospheric science phase).	Details explained wholly in above subsection introduction.

II.4 Final Gantt Chart

	Task Name	Start Date	End Date
1			
2	**Web interface**	**01/13/17**	**05/11/17**
3	Design	01/13/17	02/09/17
4	Framework	02/06/17	05/11/17
5	**Upload photo**	**03/27/17**	**04/28/17**
6	Explore dynamic upload/research	03/27/17	04/11/17
7	Implement	04/12/17	04/28/17
8	**Retrieve images**	**01/19/17**	**05/11/17**
9	Script dev	01/19/17	04/28/17
10	Script testing	04/28/17	05/11/17
11	**Wunderground**	**02/27/17**	**05/10/17**
12	Data retrieval	02/27/17	04/28/17
13	Algorithm input	04/29/17	05/10/17
14	**Display content**	**04/12/17**	**05/11/17**
15	Implement	04/12/17	05/11/17
16	**Integration**	**03/08/17**	**05/11/17**
17	**Testing of product**	**05/15/17**	**05/17/17**
18	**Code documentation**	**02/13/17**	**05/15/17**
19	Version 1.0 Beta	05/12/17	05/12/17
20	Version 1.0 Alpha	05/19/17	05/19/17

Fig. 2. Final Gantt chart requirements

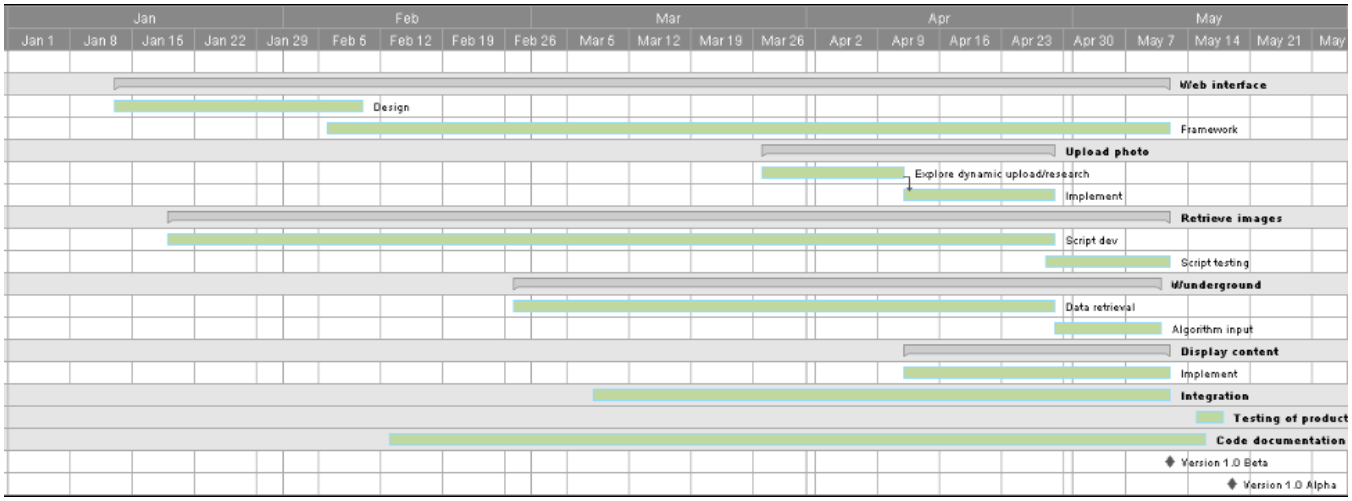


Fig. 3. Final Gantt chart timeline

III DESIGN

III.1 Original Design Document

SIGNATURES

Kim Whitehall

Client, NASA JPL

Signature: _____

Date: December 1, 2016

Lewis McGibbney

Client, NASA JPL

Signature: _____

Date: December 1, 2016

E. Reilly Collins

Student, Oregon State

Signature: _____

Date: December 1, 2016

Sophia Liu

Student, Oregon State

Signature: _____

Date: December 1, 2016

Jesse Hanson

Student, Oregon State

Signature: _____

Date: December 1, 2016

CS CAPSTONE DESIGN DOCUMENT

DECEMBER 1, 2016

AEROSOL ANALYZER MOBILE WEB APPLICATION

PREPARED FOR

NASA JPL

KIM WHITEHALL, LEWIS MCGIBBNEY

PREPARED BY

GROUP 22

AEROLYZER

E. REILLY COLLINS

SOPHIA LIU

JESSE HANSON

Abstract

Aerolyzer is a mobile web application capable of processing visible images and inferring atmospheric phenomena to provide the general public with near-real time monitoring of aerosol conditions. This document outlines the software design descriptions for the Aerolyzer mobile web application and acts as a representation of a software design to be used for communicating design information to stakeholders. A written description of the Aerolyzer software product is provided.

CONTENTS

List of Figures

List of Tables

TABLE 1
Revision history

Date	Version	Description	Author
25 Nov 2016	1.0	Initial version of document	E. Reilly Collins
26 Nov 2016	1.0	Added summary	Sophia Liu
28 Nov 2016	1.0	Added Pylint and ExifRead sections	E. Reilly Collins
30 Nov 2016	1.0	Added MySQL section	E. Reilly Collins
29 Nov 2016	1.0	Added TravisCI and Coveralls sections	Sophia Liu
30 Nov 2016	1.0	Added Sphinx and MISR sections	Jesse Hanson
30 Nov 2016	1.0	Added Wunderground and Django sections	Sophia Liu
1 Dec 2016	1.0	Added DropzoneJS section	Jesse Hanson

I INTRODUCTION

This document describes the conceptual design of the Aerolyzer mobile web application, according to guidelines presented in the IEEE Std. 1016-2009 Recommended Practice for Software Design Descriptions (SDD). The software is a web-based mobile application project that enables users to discover information about aerosol content based on uploaded images.

I.1 Purpose

The purpose of this document is to communicate design information to stakeholders for the Aerolyzer mobile web application. Software design descriptions are provided in order to give the software development team overall guidance to the projects architecture.

Furthermore, this document aims to specify the features and requirements of the Aerolyzer software. The reader of this software design document will gain a technical understanding of Aerolyzer's design and implemented functionality.

I.2 Scope

This software will be a mobile web application for use by the general public. This application will be designed to uniquely utilize color distribution within an uploaded mobile image to identify features necessary for analyzing aerosol content. The end product will be a full mobile web app that employs an open source algorithm translating how the human brain interprets images using RGB values. Citizens in the general public will be able to quantify color in images and use the resulting data to determine current aerosol content.

More specifically, users will be able to upload an image of an outdoor scene taken on a mobile phone. The application will then give output detailing the current aerosol content of the atmosphere based on the uploaded image. This software will provide average citizens with near-real time monitoring of atmospheric conditions.

I.3 Context

The task of this project is to provide the general public with near-real time monitoring of aerosol conditions via a mobile web application.The web interface is aimed to give users knowledge about current atmospheric conditions, combined with meteorological and geo-related data, based on images they upload to the app. Additionally, collected user results will form a data set that can be useful for research concerning atmospheric aerosol content.

The intended audience for this document are engineers or other stakeholders (including developers, software testers, and team managers) directly involved in the development and maintenance of the Aerolyzer mobile web application. This document is intended to provide these users with an understanding of the softwares design and underlying functionality.

I.4 Summary

The user will use this application to see the aerosol content in the air. First the user will take a picture of the skyline, and upload the unfiltered picture in the Aerolyzer application. The application will then analyze the data from the picture, and find out the time, date, and location it was taken. It will then send a request to Wunderground, and MISR, and using an algorithm, analyze the aerosol content. This data will then be re-packaged and presented to the user.

II References

[1] B. Allen. (1996) Atmospheric aerosols: What are they, and why are they so important? [Online]. Available: http://www.nasa.gov/centers/langley/news/factsheets/Aerosols.html

[2] NASA. Misr. [Online]. Available: http://www-misr.jpl.nasa.gov/

[3] Sphinx. (2016) Sphinx. [Online]. Available: http://www.sphinx-doc.org/en/1.4.8/index.html

[4] J. Castro. (2016) Review of python static analysis tools. [Online]. Available: http://blog.codacy.com/2016/01/08/review-of-python-static-analysis-tools/

[5] T. CI. (2016) Travis ci. [Online]. Available: https://travis-ci.com/

[6] L. H. Industries. (2016) Coveralls. [Online]. Available: https://coveralls.io/

[7] Pylint. Tutorial. [Online]. Available: https://pylint.readthedocs.io/en/latest/tutorial.html

[8] M. Corporation. (2016) Cve security vulnerability database. [Online]. Available: http://www.cvedetails.com/

[9] DB-Engines. Mongodb vs. mysql vs. solr comparison. [Online]. Available: http://db-engines.com/en/system/MySQL%3BSolr%3Bmongodb

[10] W. Underground. Weatherapi: Introductions. [Online]. Available: https://www.wunderground.com/weather/api/d/docs

[11] M. NASA JPL. (2016) Misr science software code available to users. [Online]. Available: http://www-misr.jpl.nasa.gov/getData/accessData/misrSoftware/

[12] J. M. K. NASA JPL. (2016) Misr order and customization tool. [Online]. Available: https://l0dup05.larc.nasa.gov/MISR/cgi-bin/MISR/main.cgi

[13] (2016) Exif tags. [Online]. Available: http://www.sno.phy.queensu.ca/~phil/exiftool/TagNames/EXIF.html

[14] P. S. F. Python Package Index. (2015) Exifread 2.1.2. [Online]. Available: https://pypi.python.org/pypi/ExifRead

[15] M. Meno. (2016) Dropzonejs. [Online]. Available: http://www.dropzonejs.com/#

[16] threespot. (2016) Django. [Online]. Available: https://www.djangoproject.com/

III Glossary

Aerosol Minute particles suspended in the atmosphere. When these particles are sufficiently large, we notice their presence as they scatter and absorb sunlight. Their scattering of sunlight can reduce visibility (haze) and redden sunrises and sunsets. [1]

EXIF/exif Exchangeable image file format specifying the specs of digital/smartphone cameras.

MISR Multi-angle Imaging SpectroRadiometer satellite. [2]

IV Design

IV.1 Identified stakeholders and design concerns

Stakeholders for this project consists of the development team and users (the general public). The development team's concerns include creating maintainable software and producing accurate results, while the users are concerned with having their images secured and safe, receiving accurate output, an easy to use mobile web interface, and quick responsiveness.

These concerns have been discussed and will be addressed in the final product by: utilizing tools for the coding environment that increase code maintainability; creating test cases that check aerosol content results against actual data to improve the core algorithm, in addition to using data sources that provide accurate image metadata and real-time atmospheric data; using a trusted and researched database management system to store user images; and combining existing tools with user experience research to create a simple, user-friendly interface that increases responsiveness.

IV.2 Design viewpoint - Resources

The purpose of the Resources viewpoint is to model the characteristics and utilization of resources in a design subject. Key concerns for our project that are addressed in the following design views are creating maintainable software (Pylint, Sphinx) and creating test cases (Travis CI and Coveralls). These concerns will be used to evaluate the effectiveness of our design regarding this specific viewpoint.

IV.2.1 Design view - Sphinx

Introduction Throughout the development of our application, our team requires the ability to create professional and informative documentation of our API documents. In order to satisfy this requirement, our team will be making use of Sphinx - a python documentation generator. This documentation tool provides a multitude of different functions including the ability for semantic markups, automatic links for citations, a built in glossary structure, and much more. [3] Moreover, this technology is compatible with Python, which is an essential requirement for our documentation process. Through its various functions, this technology will allow us to produce clear and concise documentation on our code and how to use our application. Furthermore, its output formats include HTML, LaTex, and several others, which will be useful in creating easily accessible and compatible documents for users of our application.

Design element - Sphinx functionality Sphinx provides a documentation structure that is quick and easy to use, allowing its users to connect multiple files within a hierarchy of documents. The master document serves as a welcome page and contains a table of contents which users may customize to fit their needs. [3] This ability will allow our team to develop both code and usage documentation in a way that is both professional and instructive. Moreover, it will make our documentation more readable and easily accessible to our users.

Design element - Sphinx style component In general, our team will be utilizing the default functionality and style aspects of Sphinx. Since this API is highly customizable, we plan to customize our documents in a way that follows standard IEEE guidelines. However, we will also be making use of the various tools that accompany this API, including semantic markups, glossary terms, automatic links for citations, etc. [3]

Design element - Sphinx framework For the development of our application, our team will be making use of Sphinx version 1.4.9, which is currently their newest version in production. [3] The Aerolyzer Github repository will contain the necessary code and documentation templates shortly.

Authored by Jesse Hanson

IV.2.2 Design view - Pylint

Introduction Since our team will be using Python to implement the Aerolyzer web application, we need a style guide that is Python-specific. An existing technology that would accomplish this aim is Pylint. Pylint is a tool that searches for style errors in Python code in order to enforce a coding standard. [4] One goal for the Aerolyzer project is high readability: we want our code to be easily read and understood by every member of the development team. Another goal for the use of Pylint in our design is to be able to catch syntax errors in order to avoid bugs. Python uses indentation to denote blocks, and significant logic errors can arise when whitespace is accidentally misused. We want to avoid such errors and reduce time spent on bugs that have a quick fix. A final goal for technology used in our coding environment is ensuring maintainability. The Aerolyzer software should live beyond our senior capstone class; this means other programmers will potentially need to read and add to our code. The handover and upholding process will be much smoother for our client (and result in better-quality software) if all code follows a specific format that can be learned by future team members.

Design element - Pylint subprogram functionality Pylint defines a default coding style and can also look for specified coding errors, as well as provide refactoring suggestions. The software tries to avoid false positives while attempting to detect dangerous code blocks. After analyzing the Python code, Pylint offers an overall grade for the code based on the amount and severity of warnings or errors. This tool will unify the Aerolyzer teams coding practices in order to be more efficient, as well as to create maintainable code.

Design element - Pylint Style component Specific guidelines included in our Pylint file include using 4 spaces instead of tabs for indentation, enforcing a max line length of 100 characters, and checking the import order using isort (a feature of Pylint implemented using isort). Other than these specifications, we are utilizing the default Pylint configuration (pylintrc).

Design element - Specific Pylint framework Our project will use Pylint version 1.6.4. The Aerolyzer Github repository contains our lint file, aerolyzer_lint_file. Usage for our Aerolyzer lint file is `pylint --rcfile=aerolyzer_lint_file`.

Authored by E. Reilly Collins

IV.2.3 Design view - Travis CI and Coveralls

Introduction Testing the code base while developing it is crucial to solve bugs as the code is being developed so the code does not end up with a million of bugs in the end. Since Aerolyzer is hosted on Github, TravisCI and Coveralls are the best options for our project. TravisCI is an open-source continuous integration service used to build and test software projects hosted on Github [5]. Github makes it really simple to integrate TravisCI into our project. Coveralls is a web service to help track the code coverage over time, and ensure that all of the new code is fully covered [6]. It is extremely useful to check to see what tests cover what code, and what needs to be changed to help reach as close to 100 percent coverage as possible.

One goal we have is easy integration. We want to make sure that the technology we use can be easily integrated into our code base. Both TravisCI and Coveralls is compatible with Github, and easily integratable. Another goal is keeping

our entire program open source. TravisCI and Coveralls are both open-source programs. This ensures that our program will live beyond this project, and other programmers can work on the code because it is open source.

Design element - TravisCI and Coveralls subprogram functionality TravisCI triggers automatic builds to every change to the Github code base including all branches and pull requests. TravisCI is built on the ideas of continuous integration, so basically everytime a new feature or change is added, TravisCI will run all tests to make sure that they all pass and the new code did not break some other part of the code[5]. Coveralls is a tool that goes along with TravisCI and helps gather even more information about the test that are being run. Using Coveralls there are individual file coverage reports, line by line coverage reports, and repository coverage reports [6]. It adds even more knowledge about the test coverage on top of the testing that TravisCI does.

Design element - TravisCI Build Environment The .travis.yml file customizes the build environment[5]. The file specifies that we are using Python, the versions 2.6, 2.7, 3.2, 3.3, 3.4, 3.5, 3.6, and to run coveralls at the end.

Authored by Sophia Liu

IV.2.4 Design rationale

Sphinx Overall, the justification for our decision to use Sphinx as opposed to several other documentation generators is quite simple. First and foremost, Sphinx was recommended to us by our client in light of the fact that Sphinx is compatible with our code, seeing as it is a Python documentation generator. However, several other reasons also influenced our decision to use Sphinx. The customizability of this tool was greatly appealing to us, as it will allow us to develop the documents according to our own specifications, as well as according to the requirements of our client and our senior design course. Moreover, the tools that accompany the Sphinx documentation generator will allow us to more easily create professional and informative documents that will aid in our users understanding of our application.

Pylint Justification for this decision is based on the many benefits of using Pylint discovered through dialogue with our client, as well as researching the tool's documentation. These include its ability to catch bugs ranging from small to serious, more easily provide our team with a unified coding style, and offer refactoring features. Furthermore, Pylint is freely available, currently maintained with no evidence of becoming obsolete [4], and has plenty of easily understood documentation. [7]

Our team has spent some time in our development cycle to narrow down a coding style for the Aerolyzer project and configure our own lint file. The time we would spend completing a code analysis with Pylint is definitely not equal to the time our team would otherwise spend manually finding syntax and logic errors. This time tradeoff will be especially relevant for any team members unfamiliar with coding in Python.

TravisCI and Coveralls Justification for this decision is based off of the goals and the parameters of our project. Since we are using Github, TravisCI and Coveralls are the best option for testing. They are easily integrated into Github which makes them very easy to use to our project. They both have fairly new versions, and are being well-maintained. Another great thing about these two tools is that they are both completely open-source which works great for our project [5]. This aligns with one of our main goals to keep our project open-source so other people can eventually contribute and use our program.

IV.3 Design viewpoint - Dependency

IV.3.1 Design view - MySQL

Introduction The Aerolyzer system will required use of a database management system to perform data retrieval through a query language, securely store data, allow for concurrent access by multiple users, and keep data backed-up. The DBMS we have selected is MySQL. The maintained storage will consist of uploaded landscape images and their associated metadata, meteorological data, and geo-related data. Access to this database is necessary for a range of users, including members of the Aerolyzer development team (for improving the core algorithm) and any researchers who would like to make use of our collected data. The database will need to be backed-up regularly in the event of data loss or other situation that required data recovery.

Design element - MySQL data store The database design is illustrated in Figure 1. It is divided into four entities: a table for image data, acquired via EXIF metadata extraction; a table for geo-related data, acquired via MISR; a table for weather data, acquired from Weather Underground; and a table for the core algorithms output, acquired after running the core algorithm successfully and receiving output. The primary key that relates an image to its associated geo, weather, and aerosol data is `image_id`. This simple setup and division into four tables was a design decision made based on the desired functionality of querying different data sets: queries related to uploaded images, weather data, satellite data, or the atmospheric conditions is separated to achieve a cohesive framework.

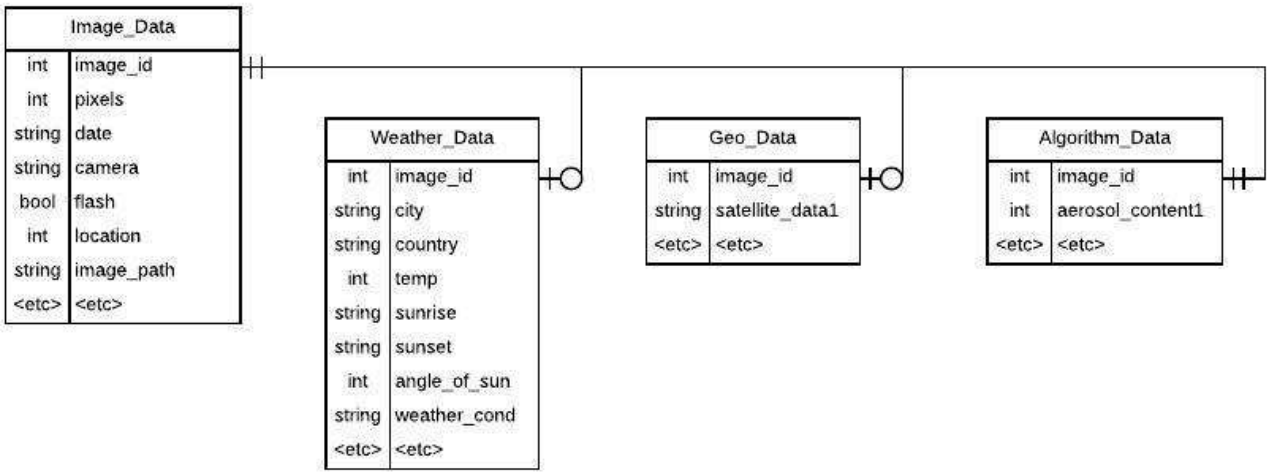


Fig. 1. MySQL data store design

Design element - Image_Data entity The image data entity will store image EXIF metadata. Currently, we are in the research phase of developing the core color-analyzing algorithm and therefore have not determined what specific EXIF metadata will be required. The specific database rows will be updated as the design of the algorithm is finalized. The `image_id` is a primary key for the image across all entities. In addition, certain image metadata will be required as input for the core algorithm in order to produce results. Therefore, this entity is required.

Design element - Geo_Data entity Because we have not completed the design for the core algorithm, the data that will be pulled from the MISR source is still unknown. However, the specific rows containing geo-related data will be

updated once the algorithm is designed. This entity is optional, as satellite data from MISR may not be available based on the images location.

Design element - Weather_Data entity Required meteorological information for the core algorithm is more concrete at this point in the teams research. Since we will be making use of the Weather API from Weather Underground, the team was able to determine what weather-related information is able to be extracted based on location information pulled from EXIF metadata. If other meteorological information is needed once the core algorithm has been designed, however, more rows will be added to this table. This entity is optional, as weather data from Weather Underground may not be available based on the images location.

As shown in Figure 1, data for a specific image will be stored in multiple tables based on information extracted from different sources. The workflow will be similar to the following:

- SubjectLocation EXIF tag will be pulled from an uploaded images metadata
- Location is stored in the `Image_Data` table
- This location is used to determine the specific city and country the image was taken in via the Weather API
- This city and country information is stored in the `Weather_Data` table.

Design element - Algorithm_Data entity The core algorithm will use data from the other three entities as input and output aerosol content information. Currently, the core color-analyzing algorithm is being researched by the client stakeholder; as a result, the output of the algorithm (beyond the percentage of aerosol content) relating to atmospheric conditions is still unknown. Specific rows correlating to the output of the algorithm will be added to the database as needed. This entity is required, as any images that are stored in the `Image_Date` table will have associated algorithm output that needs to be documented.

Authored by E. Reilly Collins

IV.3.2 Design rationale

The Aerolyzer application will need a centralized storing location for images, associated metadata, and output from the color-analyzing core algorithm. Although we could make use of something simple, such as files in a shared directory, a database would be best suited for our needs: being able to maintain a repository of collected images and corresponding aerosol information will allow us to build up data for use in research, compare users results against real aerosol content, and query the data set for specific information that would be useful for our team and the research community.

The deciding factor for using MySQL as a database management system is security. MySQL has not had any discovered security vulnerabilities in 2016. [8] The system was initially released in 1995, meaning it has stood the test of time and benefitted from years of continued improvement. [9] Additionally, MySQL utilizes a fine-grained authorization concept allow us to further secure our data and tailor user accessibility to the security needs of our system. There is also support for making data persistent, which will be useful for our needs as we will likely not modify the images and associated metadata. Lastly, this DBMS is free and currently maintained.

IV.4 Design viewpoint - Composition

The purpose of the Composition viewpoint is to describe the way the design subject is (recursively) structured into constituent parts and establish the roles of those parts. Key concerns for our project that are addressed in the following

design views are obtaining accurate results both for the user and research purposes for the development team by using data sources that provide accurate image metadata and real-time atmospheric data. These concerns will be used to evaluate the effectiveness of our design regarding this specific viewpoint.

IV.4.1 Design view - Weather Underground

Introduction In order to accurately analyze aerosol content in the pictures, it is important to have accurate real-time meteorological data. When users submit their pictures,the location and time is extracted . In order to get accurate aerosol information, the weather information is needed. The Weather Underground (Wunderground) API is the best tool to do this. The Weather Underground provides real-time temperature, air pressure, clouds, wind chill, dew point, humidity, and rainfall. They also provide radar maps. The radar maps can be either static or animated, depending on the call. The latitude and longitude is sent to the API, and it will return a radar map based on the call. [10]

Design element - Weather Underground subprogram functionality To use the API, the first required thing is to get an API key. Weather Underground API takes in a HTTP request, and returns either JSON or XML. We will then extract the data we need and use the information to help derive the Aerosol content in the air. [10]

Design element - Specific Weather Underground Plan To make our requests economical, we will be using the Stratus plan. The Stratus plan only returns the information we need, so we do not need to sort through a large amount of data. This plan returns geolookup, autocomplete, current conditions, 3-day forecast, astronomy, and almanac for the day [10]. To make the request even more efficient, we can specify the exact information we need, not just every thing that is returned in the Stratus plan. This will decrease the processing time from when the user uploads the photo, to when the user is returned the aerosol information based on their picture.

Authored by Sophia Liu

IV.4.2 Design view - MISR

Introduction Seeing as the goal of our application is to accurately analyze the aerosol content of various images, we need the ability to obtain reliable and consistent geo-related data. When searching for a technology that could satisfy this requirement, we focused on a few essential features. We need an API that is capable of obtaining truly accurate information on aerosol content, cloud formation, and light reflection. Moreover, this technology will need to be highly reliable and capable of continually supplying data, otherwise the accuracy of our results will suffer.

Therefore, to accomplish the task of obtaining geo-related data, we will be making use of NASAs Multi-angle Imaging SpectroRadiometer (MISR). For all intents and purposes, MISR is an API which provides continual global coverage of the sunlit Earth in high spatial detail via nine widely spaced angles. [2]

Ultimately, we chose MISR as our technology for this task for a multitude of reasons. First and foremost, as a technology used by NASA, the accuracy of the data being provided will almost certainly be reliable. Along with this however, it is typically safe to say that the funding behind this API will be reliable enough to guarantee continual coverage.

Design element - MISR data extraction To gain access to the data provided by MISR, we must meet a minimum set of requirements according to NASA JPL. These requirements include EOSDIS Earthdata Login account, being CSS and Javascript enabled, and having HTML cookies enabled. Once these requirements have been met, we will be able to

make use of MISR visualization and analysis tools made available via MISRs IDL data processing software package. This software package will include all of the necessary data processing software provided by the Atmospheric Sciences Data Center (ASDC) in production of data from MISR that is level 1 or higher. Moreover, it includes other tools such as data visualization and analysis used by MISR. [11]

Design element - MISR data customization Once we have access to the data provided by MISR, we will be able to uniquely order and customize the data we receive via MISRs Order and Data Customization Tool. [12] This will allow us to order customized level 2 data from MISR, as well as quickly download non-customized level 2 and level 3 MISR data. When used correctly, this tool will allow us to obtain more refined data according to specific dates, times, locations, etc.

Authored by Jesse Hanson

IV.4.3 Design view - ExifRead

Introduction In order to ascertain aerosol content, the core color-analyzing algorithm needs an image's metadata as input. We are interested specifically in EXIF metadata that will provide the algorithm with needed information, such as the type of smartphone the image was taken on and the location of the photo. Having a wide range of information about uploaded images for the algorithm's input will in turn give output that is as accurate as possible. Our main goal regarding EXIF metadata extraction is to pull as much data from the image as possible. We will be making use of ExifRead, a technology that extracts all available EXIF data from an image, allowing us to use that data as input for the core algorithm.

Design element - Required data for EXIF object Currently, we are in the research phase of developing the core color-analyzing algorithm and therefore have not determined what specific EXIF metadata will be required. However, ExifRead is able to provide all standard EXIF metadata, so we can extract data corresponding to any tags we end up needing. An example of potential EXIF header data is listed in the UML diagram in Figure 2. [13] This EXIF data will be pulled from uploaded images using the ExifRead package and stored in an object for use as input in the core algorithm.

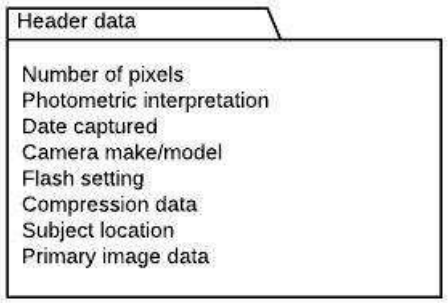


Fig. 2. Example EXIF metadata

Design element - EXIF extraction function After installing the ExifRead Python package, the extraction of EXIF metadata will be accomplished via a function that stores the extracted EXIF data in an object using key-value pairs.

This object can then be used as input for the core algorithm, and values can be accessed using keys as strings indicating the header data type using standard EXIF tags. This function will behave similar to the following code and result in an object similar to Figure 2: [14]

```
import exifread

# f: image to be processed
# Exif_Data: tuple to store EXIF data for current image
def get_exit_data(f, Exif_Data):

# Return EXIF tags
tags = exifread.process_file(f)

# Loop through tags and store required data in an object
for tag in tags.keys():
    if tag in ('SubjectLocation', 'PhotometricInterpretation', <etc>):
        cur_exif_data = Exif_Data(tag = tags[tag])
```

Authored by E. Reilly Collins

IV.4.4 Design view - DropzoneJS

Introduction Since our application will ultimately allow users to analyze the aerosol content of various pictures, our web application requires the ability for users to upload images. Our requirements for this technology are very simple. The API should be easy to use, while also allowing users to upload high quality images when may then analyze for aerosol content. We emphasis the necessity of high quality images because without this constraint, we would not be able to adequately analyze the content of the image being uploaded. Moreover, we need a technology that allows users to upload images both via a mobile device and from the web, and furthermore must be compatible with our code.

To satisfy this requirement, we decided to go with the DropzoneJS. DropzoneJS is an open source library which does not rely on other libraries and is highly customizable. Moreover, it provides a simple drag and drop file upload and can be used programmatically. Finally, this API is ideal because it is compatible with most common browsers, including Chrome, Firefox, Internet Explorer (IE), and Safari. [15]

Design element - DropzoneJS dropzone creation In order to begin uploading files, DropzoneJS first requires that a dropzone be created which can accept the file to be uploaded. There are a couple of ways in which we can go about this. Most commonly, dropzones are created as a form element via use of the class dropzone. From there, DropzoneJS will upload to that dropzone all images which are dropped into the action attribute of that form element. Alternatively, dropzones can be created programmatically by instantiating the Dropzone class, giving us some versatility in our implementation. [15]

Design element - DropzoneJS drag and drop functionality One useful feature of DropzoneJS is its drag and drop functionality which allows users to drag and drop file directly into our web application for upload. This functionality

can be set up programmatically; however, it is only supported recent versions of the most common browsers, including Firefox, IE, Chrome, Safari, etc. [15]

Design element - DropzoneJS server side implementation While DropzoneJS is simple, easy to use, and provides us with convenient dropzone creation and drag and drop functionality, it does not have server side implementation to determine how the uploaded files are handled. However, the file upload process is similar to that of a simple form file upload, therefore we can make use of several different server side implementations such as PHP to handle our files. [15]

Authored by Jesse Hanson

IV.4.5 Design rationale

Weather Underground Weather Underground is an accurate way to efficiently gather real-time meteorological data. The API is set up so data can be easily obtained. The only requirements are the location, which every picture will have since they are all taken on mobile devices. It also aligns with one of our big goals which is to keep this project open-source and free. Wunderground API is a free service so other developers can work on or use our code base.

MISR Ultimately, MISR is the best technology available to us in terms of obtaining geo-related data for several reasons. As previously mentioned, since MISR is controlled by NASA JPL, the data provided by this technology will be both highly reliable and consistently provided. More specifically however, the data processing software packages provided by MISR and ASDC will be greatly beneficial to us as they will allow us to efficiently order and customize the data we receive. Along with this, we will have access to data visualization and analysis tools used by MISR, which will come in handy as we attempt to use this geo-related data to accurately analyze the aerosol content of various images.

ExifRead The ExifRead package supports various versions of Python and returns GPS and manufacturer data in addition to standard EXIF metadata. [14] GPS information will provide the core algorithm with additional meteorological data based on the image's location, and manufacturer data will be helpful for narrowing down the camera specs. It will be easy to use within our code, as it is a Python package, and we will not have to worry about storing our images in multiple locations: EXIF metadata extraction can be done within the code rather than via an uploaded image. Lastly, as demonstrated in Figure 2, ExifRead will provide all of the metadata currently required of the Aerolyzer application.

DropzoneJS While there are several available technologies for uploading files, we chose to go with DropzoneJS because its functionality best fit our applications needs. The versatility of DropzoneJS in its creation of dropzones provides us with a couple of options in our development. Moreover, one of our criteria was to have a simple and easy to use file upload, which is made possible by DropzoneJS via its drag and drop functionality. Lastly, while we will have to develop our own server side implementation to determine how images are handled once uploaded, DropzoneJS is compatible with our code and will serve us well in making the first step of our application easily usable for our users.

IV.5 Design viewpoint - Interface

The purpose of the Interface viewpoint is to provide information for designers, programmers, and testers to know how to correctly use the services provided by a design subject. This section addresses design concerns of creating a simple, user-friendly interface that increases responsiveness. These concerns will be used to evaluate the effectiveness of our design regarding this specific viewpoint.

IV.5.1 Design view - Django

Introduction Frameworks are an invaluable resource for developing web applications. The purpose of a framework is to allow the application to be well structured, and allows for other developers to easily maintain or upgrade the application. The goal should allow for the developers to not have to focus on low-level details. The framework should also be a fullstack framework. A fullstack framework is a framework that covers all layers of the stack, this essentially means that it will help the application be developed more efficiently without having to develop any layer from scratch. [16] Django is a high-level Python Web framework that ”encourages rapid development and clean, pragmatic design.” [16] Django also has a lot of built in security features, web forms, and templates.

Design element - Django subprogram functionality Django has built in features to help with database transactions and manipulations. This will be useful when we are inserting pictures and data into the database. Django also has a really nice templating system. These systems auto generate HTML to make it simpler to create web pages. These pages can still be changed to fit the needs of the application. We will need at least 2 pages, one for uploading the picture, and one for displaying the data. A very important feature of Django is its security features. Since users are uploading personal photos, security is very important because users will not upload personal information into an application that is not secure. Django has cross-site scripting protection, Cross site request forgery protection, SQL injection protection , Clickjacking protection, and Host header validation. [16] Security is a concern for many people, and Django is great at having a lot of built-in security features.

Authored by Sophia Liu

IV.5.2 Design rationale

Django Django also has a lot of built in security features, database manipulations, and templates. Security is key to make sure that nobodys personal information or pictures gets taken. People will not use an application that has security flaws. The templating system is also very useful. Writing every HTML page from scratch is a lot of unnecessary work, so having the pages auto-generated is time-saving. The database manipulation will make inserting, deleting, and editing our SQL database a lot easier. Frameworks are designed to make developing easier, and Django does just that.

IV.6 Design viewpoint - Algorithm

The purpose of the Algorithm design viewpoint is to provide a detailed design description of operations (such as methods and functions), the internal details and logic of each design entity. This section addresses design concerns of the speed and accuracy of the core color-analyzing algorithm. These concerns will be used to evaluate the effectiveness of our design regarding this specific viewpoint.

IV.6.1 Design view - Core color-analyzing algorithm

Currently, the core color-analyzing algorithm is in the research and development stage being performed by the client stakeholder. The design and testing for this core algorithm is not available to the editors of this SDD, and therefore the design concerns of its speed and accuracy cannot be described in this document.

V Requirements Matrix

TABLE 2
Functional requirements in SRS and SDD

SRS ref	Functional requirement	SDS ref	Design viewpoint
III-B-1	Web Application Interface	IV-E	Interface
III-B-2	Upload Photo	IV-E	Interface
III-B-3	Display of Aerosol Content	IV-C	Dependency
III-B-4	Extract EXIF Information From Data	IV-D	Composition
III-B-5	Using Weatherunderground API	IV-D	Composition
III-B-6	Get Data From Satellite Source	IV-D	Composition
III-B-7	Running the Core Algorithm	IV-F	Algorithm

III.2 Design Document Changes

The design document did not have any large changes over the course of the year. The only change our team made was to change the database schema to have three tables instead of four tables. The original schema had a table for image data, acquired via EXIF metadata extraction; a table for geo-related data, acquired via MISR; a table for weather data, acquired from Weather Underground; and a table for the core algorithm's output, acquired after running the core algorithm successfully and receiving output. Our new schema has the images table, a table for image data acquired via EXIF metadata extraction, geo-related data acquired via MISR, weather data acquired from Weather Underground, the core algorithm's output acquired after running the core algorithm successfully and receiving output

IV Tech Review

IV.1 Original Tech Review

Technology Review and Implementation Plan

Aerolyzer

Group 22

E. Reilly Collins, Sophia Liu, Jesse Hanson

CS 461 Fall 2016

November 14, 2016

Abstract

This document examines potential technologies for use in different pieces of the Aerolyzer project. Technology for each piece is narrowed down to three choices based on the goals of our project, then the three options are compared in more detail using predetermined criteria. After conducting research and analyzing trade-offs or pointing out features that offer extra benefits, the selected technology is identified for each piece. An explanation is given based on the unbiased and well-considered analysis.

CONTENTS

List of Tables

I. Introduction

Aerolyzer is a mobile web application capable of processing visible images and inferring atmospheric phenomena to provide the general public with near-real time monitoring of aerosol conditions. In this document, we will be focused on determining which technologies will be used throughout the duration of this project. The functions for which we need technologies are as follows: Coding environment, image and metadata storage, extract EXIF metadata, web interface, getting meteorological data, testing, upload photos, getting geo-related data, documentation/development of API documents.

E. Reilly Collins: Technologies 1, 2, and 3

Sophia Liu: Technologies 4, 5, and 6

Jesse Hanson: Technologies 7, 8, and 9

II. Coding Environment

A. pyLint

Since our team will be using Python to implement the Aerolyzer web application, we need a style guide that is Python-specific. One such existing technology that would accomplish this aim is Pylint. Pylint is a tool that searches for style errors in Python code in order to enforce a coding standard. [1] It defines a default coding style and can also look for specified coding errors, as well as provide refactoring suggestions. The software tries to avoid false positives while attempting to detect dangerous code blocks. After analyzing the Python code, Pylint offers an overall grade for the code based on the amount and severity of warnings or errors. Pylint is not perfect solution for enforcing a cohesive coding standard it can be too verbose with warnings and slow with its analysis. [2] However, Pylint is certainly an option for unifying the Aerolyzer teams coding practices in order to be more efficient.

B. Goals

With regards to our coding environment, we are mainly striving for readability: we want our code to be easily read and understood by every member of the development team. Another goal for the use of Pylint in our design is to be able to catch syntax errors in order to avoid bugs. Python uses indentation to denote blocks, and significant logic errors can arise when whitespace is accidentally misused. We want to avoid such errors and reduce time spent on bugs that have a quick fix. A final goal for technology used in our coding environment is ensuring maintainability. The Aerolyzer software should live beyond our senior capstone class; this means other programmers will potentially need to read and add to our code. The handover and upholding process will be much smoother for our client (and result in better-quality software) if all code follows a specific format that can be learned by future team members.

C. Criteria

The merits of using technology for our coding environment will be evaluated based on several factors. These include cost, availability, and speed. In order to keep our costs for this project at zero, we need to take advantage of technology that is free - preferably something that is open source. Along the same lines, we want to use technology that will be available and maintained long-term. If our coding environment software becomes deprecated or defunct, our own Aerolyzer software will have to spend time searching for and incorporating a different technology (which is something we definitely want to avoid). Finally, the speed of technology used in our coding environment is an important feature: it would be pointless to exchange time spent finding errors manually for running slow tools that check for errors.

D. Selection

In discussions prior to this Technology Review, our team and clients had already made the decision to make use of Pylint for enforcing a Python coding standard within our project. Therefore, as indicated by our professor, we did not need to research other coding environment technologies - we have presently begun moving forward with a Pylint configuration based on our clients setup and previous experience. Justification for this decision is based on the many benefits of using Pylint discovered through dialogue with our client, as well as my own research. These include its ability to catch bugs ranging from small to serious, more easily provide our team with a unified coding style, and offer refactoring features. Furthermore, Pylint is freely available, currently maintained with no evidence of becoming obsolete [3], and has plenty of easily understood documentation. [4]

No tool is without flaws: a common complaint with the Pylint software is that noisy output can be both unhelpful and cause slow analysis. [2] However, Pylint is configurable, so our team can spend some time at the beginning of our development to narrow down a coding style for the Aerolyzer project. This discussion will allow us to end up with a Pylint configuration that reduces unnecessary output and increases analysis speed. Additionally, the time we would spend completing a somewhat slow code analysis is definitely not equal to the time our team would otherwise spend manually finding syntax and logic errors. This time tradeoff will be especially relevant for our student team members - we do not have as much experience coding in Python.

III. Image and Metadata Storage

A. Available Technologies

The Aerolyzer application will need a centralized storing location for images, associated metadata, and output from the color-analyzing core algorithm. Although we could make use of something simple, such as files in a shared directory, a database would be best suited for our needs: being able to maintain a repository of collected images and corresponding aerosol information will allow us to build up data for use in research, compare users results against real aerosol content, and query the data set for specific information that would be useful for our team and the research community. Three database options we decided to research are Apache Solr, MongoDB, and MySQL. Apache Solr is a standalone search server with a REST-like API [5]; MongoDB is a NoSQL database [6]; and MySQL is a SQL database management system[7]. All of these technologies are open source and support Python and concurrency. [8]

B. Goals

The benefits of using a database management system include data retrieval through a query language, secure data storage, concurrent access by multiple users, and the ability to have data backed-up or recovered. We require a maintained storage of uploaded landscape images and their associated metadata, meteorological data, and geo-related data. Access to this database is necessary for a range of users, including members of the Aerolyzer development team (for improving the core algorithm) and any researchers who would like to make use of our collected data. This use case outlines two goals for our project: being able to query our data set while allowing for multiple users to add to or pull from our database concurrently. Additionally, using a DBMS will keep our data stored more securely than it would be on a file system - this aim is significant because we want to be able to promise our users that their images will be safely contained. Along the

lines of security, a final goal involves being able to back up our dataset. Allowing for recovery in the event of data loss or some other issue will ensure that our collected information can be retained for the life of the application.

C. Criteria

The criteria that will be used to evaluate and compare the three DBMS technologies are cost, availability, and security. We want to keep our costs for Aerolyzer at zero, so making use of free software is extremely important. We also need technology that is available long-term and maintained; this is especially significant for our database technology, as we could experience a complete loss of our data if the system becomes defunct. Finally, technologies will be judged on whether data will be stored securely; as mentioned previously, our team wants to promise users that their personal images will be safe and only accessed by authorized users.

Table I compares the three technologies based on the described criteria. [8] [9]

TABLE I
IMAGE AND METADATA STORAGE TECHNOLOGIES COMPARED

	Cost	Availability	Security	Other
Solr	Free	Currently maintained, initially released in 2004, support for making data persistent	3 vulnerabilities reported in 2016	RESTful HTTP API
MySQL	Free	Currently maintained, initially released in 1995, support for making data persistent	Users with fine-grained authorization concept, susceptible to SQL injection-type attacks, 0 vulnerabilities reported in 2016	
MongoDB	Free	Currently maintained, initially released in 2004, support for making data persistent	Access rights for users and roles, JSON documents can be altered maliciously, 1 vulnerability reported in 2016	Proprietary protocol using JSON

D. Selection

Based on research revealed in the above table, we will be using a MySQL database management system. Though the three technologies are very different, they each have similar features based on specific criteria needed for Aerolyzer's data storage. The deciding factor for our purposes is security. Comparing security vulnerabilities, MySQL has not had any discovered issues in 2016. Although SQL injection-type attacks are possible when using a SQL database (MongoDB and Solr are not SQL implementations), there are steps developers can take to independently prevent these. Lastly, MySQL utilizes a fine-grained authorization concept allow us to further secure our data and tailor user accessibility to the security needs of our system.

IV. EXIF METADATA EXTRACTION

A. Available Technologies

In order to ascertain aerosol content, the core color-analyzing algorithm needs an image's metadata as input. We are interested specifically in metadata that will provide the algorithm with needed information, such as the type of smartphone the image was taken on and the location of the photo. Having a wide range of information about uploaded images for the algorithm's input will in turn give output that is as accurate as possible. The three technologies that will be compared for

this functionality are the Cloudinary Admin API, ExifRead package, and Pillow ExifTags module. All three APIs can be used in Python. [10] [11] [12]

B. Goals

Our main goal regarding EXIF metadata extraction is to pull as much data from the image as possible. Ideally, we would use a technology that extracts specific data that we need for the core algorithm's input; however, at this stage in the project, we do not have concrete metadata specifications. For example, after the client completes research for and begins designing the core algorithm, it may be discovered that the use of flash when taking a picture is significant; in that case, we would need to pull flash metadata from the image. As a result, we require a technology that extracts all EXIF metadata to aid in our core algorithm implementation.

C. Criteria

Criteria used to analyze the merits of these three technologies will be cost, availability, and what EXIF is returned. As stated throughout this document, we are attempting to create Aerolyzer at no cost; therefore, the selected API needs to be freely available. Along with free availability, a well-maintained API that will be available long-term is a significant factor in our technology selection. The chosen API should be available for as many calls as our system needs. Next, in alignment with our goals, we require technology that returns all associated EXIF metadata; furthermore, any additional photo metadata that the API returns could potentially improve our algorithm's accuracy. The availability of documentation outlining the API's capabilities will provide this information.

Table II compares the three technologies based on the described criteria. [10] [11] [12]

TABLE II
EXIF METADATA EXTRACTION TECHNOLOGIES COMPARED

	Cost	Availability	Metadata returned
Cloudinary	Free for up to 500 requests per hour	Currently maintained, limited to uploaded photos	EXIF, facial recognition, dominant color values and 32 leading colors
ExifRead	Free Python package	Currently maintained, recently updated for Python 3.4 with backwards compatibility (v. 2.6, 2.7, 3.2, 3.3)	EXIF, GPS, and manufacturer data
Pillow	Free Python module	Currently maintained	Well-known EXIF data

D. Selection

Comparing the technologies in the above table reveals various strengths for each API. The Cloudinary Admin API provides not only EXIF data, but facial recognition coordinates, dominant color values, and the 32 leading colors in the image. [10] This information would be extremely useful for our color-analyzing core algorithm. On the other hand, the ExifRead package supports various versions of Python and returns GPS and manufacturer data in addition to EXIF metadata. [11] GPS information would be provide the core algorithm with additional meteorological data based on the image's location, and manufacturer data would be helpful for narrowing down the camera specs. Some of the tradeoffs of

these APIs, however, include Cloudinary's requirement that images be uploaded to their cloud server in order for the API to return metadata. This collides with our application's use of its own database for storing images, and we do not want to store images twice. Additionally, the Pillow ExifTags module's documentation states that it return "well-known EXIF data" and is not clear on whether this includes all data or a specific subset. [12] There are no other outstanding benefits to using the Pillow implementation. For these reasons, we will select the ExifRead package to implement our EXIF metadata extraction. It will be easy to use within our code, as it is a Python package, and we will not have to worry about storing our images in multiple locations. Furthermore, we will be able to have GPS and manufacturer data for our algorithm as well as EXIF metadata.

V. Web interface

A. Available Technologies

Since the Aerolyzer application is going to be a web application, having a solid web framework is key to a successful application. Our team is using Python to implement the Aerolyzer web application which makes Django the first option. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design[13]. Our second option is Flask, flask is a microframework for Python that is based off of Werkzeug, and Jinja 2[14]. Our third option is web2py, web2py is a open source full-stack web framework that allows the fast, scalable, secure, and portable database-driven web-based applications[15].

B. Goals

The purpose of a framework is to allow the application to be well structured, and allows for other developers to easily maintain or upgrade the application. The goal should allow for the developers to not have to focus on low-level details. The framework should also support numerous activities such as getting form parameters, handling cookies, producing responses, and storing data persistently[16]. We also want the framework to be compatible with Python since that is what our application will be using.

C. Criteria

The criteria that will be used to evaluate and compare the three frameworks are cost, availability, and available tools. Since our project is open-source, we want to make sure that there will be no costs to users or developers that want to use our application. We want to make sure that everyone has access to our project, and that the framework has consistent updates and is maintained in the long-term so our project does not get stuck on a dead framework. Finally we want the framework we use to have the tools we need to develop or web application.

Table III compares the three technologies based on the described criteria. [16]

D. Selection

After reviewing our goals, the pros and cons of each framework, and the table above, we will be using Django. A fullstack framework would be more useful for our application so that rules out Flask. Django and web2py were very similar, but Django has more little updates so it is being maintained, but less big updates so the project would not have huge updates occurring too often. Django also support large companies that include pinterest and instagram that have a lot of image loading, so it will be able to support our needs of uploading and displaying images.

TABLE III
WEB INTERFACE TECHNOLOGIES COMPARED

	Cost	Availability	Tools
Django	Free	Latest update date is 2016-09-01	FullStack, has security features, scalable to large projects, caching framework, design your own templates.
Flask	Free	Latest update date is 2016-05-29	Microframework, built-in development server and debugger, integrated unit testing support, supports cookies, RESTful request dispatching
web2py	Free	Latest update date is 2016-05-10	Fullstack, MVC model, works with MYSQL, capable of uploading/downloading large files, emphasis on backward compatibility

VI. GETTING METEOROLOGICAL DATA

A. Wunderground

A large aspect of Aerolyzer is analyzing the Aerosol content in the pictures, and a crucial part of that is getting meteorological data. When users submit their pictures, they will also be submitting the location and time that will be extracted from the photo information. In order to get accurate aerosol information, the weather information is needed. The Weather Underground (Wunderground) API is the best tool to do this.

B. Goals

Our goal is to gather meteorological data efficiently, and accurately. We want our processing time once the user uploads a picture to be short, but we also want it to be accurate. These two goals are crucial for the success of our application because in this day and age, users have high expectations for applications to run fast, and without accurate meteorological data the results returned to the user will be false.

C. Criteria

The criteria of our technology forgetting meteorological data will be evaluated based on 3 factors. These include cost, availability, and speed. Since our project is open-source the tool we use must be free. It must also be available and maintained in the long-term, we do not want to use a tool that will not be available in a few years. The tool must also be fast. The speed of the application depends on the speed that the tool can send us the meteorological data, and if that is slow, then our entire application will be slow.

D. Selection

In discussions prior to this Technology Review, our team and clients had already made the decision to make use of Weather Underground. Wunderground is the only API that is free, fast and efficient. While there are some other similar APIs, Wunderground can process the most calls for free. However it still does have the limiting factor that it can only process 500 photos a day, and 10 calls per minutes [17]. Our client uses this API in her daily work and highly

recommended it, so to our team that is a great justification to use this tool. We also did our own research and concluded that Wunderground API is the best tool for our application.

VII. Testing

A. Travis CI and Coveralls

An important part of developing an application is testing. Making sure that all of your code is being covered by tests is really important to avoid bugs in the future. Since we are using Github, the best tools to do this are Travis CI and Coveralls. Travis CI is an open-source integration system that is used to build and test projects in GitHub [18]. Coveralls is an open-source tool that shows what parts of the code isnt covered by the test suite [19].

B. Goals

With goals for our testing, we want a tool that will run all of our tests. This is important because as we add code to our code base, we want to make sure that our tests are also changing to cover the code. To help with that we want to see what tests cover what part of the code, and what code is uncovered. That way we know what part of the test suite to alter to get to as close as 100 percent of the covered code as we can. This helps reduce the amount of bugs we will have to later fix. It will also help reduce the amount of unexpected errors we have later in the project when our code base is big.

C. Criteria

The criteria for the tools used for testing are cost, availability, and speed. Our project is open source so we want to make sure that the tools we use are either open-source or free. We also want to make sure that these tools will be available for long-term use. It is important to look and see how often the tools are getting updates, and how long they have been around for. One of the most important criterias is speed. When the code base gets large, the tool has to be able to keep up with the amount of tests and code. The tests have to be run fast and efficient, or else it could take a really long time to test things once the projects gets large.

D. Selection

Since we are using Github, TravisCI and Coveralls are the best option for testing. They are easily integrated into Github which makes them very easy to use to our project. They both have fairly new versions, and are being well-maintained. Another great thing about these two tools is that they are both completely open-source which works great for our project. Our client recommended these tools to us prior to this tech document, and after much research we agree that these are the best tools to integrate into our project on Github.

VIII. Uploading Photos

A. Available Technologies

In order for our application to properly analyze images provided by a user, we will need to make use of an API that allows users to upload images to our web application. This is a crucial part of our application because without it, we would have no way of retrieving images and would therefore be unable to properly analyze a users image. To fulfill this requirement, we examined three different image upload APIs that provide a multitude of different advantages. These APIs

include DropzoneJS, Ospry, and Cloudinary. DropzoneJS is an open source library which provides its users with drag and drop image uploads as well as image previews[20]. Ospry on the other hand is a very simple way for developers to upload images to the cloud while providing additional tools which makes manipulation of the image very easy[21]. Lastly, Cloudinary provides image management in the cloud, and provides end-to-end solutions for web and mobile developers[22].

B. Goals

The upload image functionality of our application should be simple and easy to use. However, it should simultaneously allow users to upload high quality images in order for us to adequately analyze the content of the image being uploaded. We need an API for image uploading that is compatible with our code, and allows users to upload images from both the web and via a mobile device.

Table IV compares the three technologies based on the described criteria. [20] [22] [21] [23]

TABLE IV
UPLOAD PHOTOS TECHNOLOGIES COMPARED

	Cost	Availability	Tools/Advantages
DropzoneJS	Free	Currently maintained	Drag and drop, open source, highly customizable, simple usage, programmatic dropzones, compatible
Ospry	Not Free	Currently maintained	Reliable cloud storage, image manipulation
Cloudinary	Not Free	Currently maintained	Reliable cloud storage, security, image manipulation

C. Criteria

When searching for an API which we will use for image uploading, we must keep in mind certain criteria. First of all, we want to ensure that uploading the image is a simple process for the user, and can be performed from both the web and via a mobile device. Moreover, we need to make sure that the API being used allows for the upload of large images, and adequately maintains all of the images data. This is a crucial aspect of this technology because we need to be able to accurately analyze the data provided by the image if we wish to determine the aerosol content from its features.

D. Selection

After extensive research on the aforementioned available technologies, our Aerolyzer application will make use of the DropzoneJS API for various reasons. First and foremost, DropzoneJS does not depend on any outside libraries and is highly customizable, which means we can manipulate it to fit our specific needs. Moreover, while DropzoneJS does not provide the server side implementation of handling the images, its does provide a simplistic file upload and allows for file dropzones to be created programmatically. This API is also compatible with most main browsers including Chrome, IE, Firefox, and Safari[20]. While Ospry and Cloudinary also provided desirable features such as reliable cloud storage and security, we chose to go with DropzoneJS because we prefered the drag and drop functionality and favored the fact that it was an open source technology.

IX. Getting Geo-related Data

A. MISR

As part of our application, our team requires the ability to obtain geo-related data in order to accurately analyze the aerosol content of the air. In order to do that, we will be using the Multi-Angle Imaging SpectroRadiometer (MISR). MISR is an API which views the sunlit Earth using nine widely spaced angles, and provides ongoing global coverage with high spatial detail. It is calibrated to provide accurate measurements of the brightness, contrast, and color of reflected sunlight[24]. The data and information provided by this API will allow us to adequately interpret the aerosol content of images sent through our algorithm by our users.

B. Goals

The goal of using a geo-related tool is to obtain accurate information of the aerosol in the air, the cloud formation, and the light reflection. Through the use of such information, we will be able to accurately analyze the aerosol content in the picture using the data we obtain from our geo-related tool. Therefore, our goal is to use an API which provides accurate information and data in large enough amounts that our aerosol content algorithm will be capable of determining the aerosol content of any given accepted picture.

C. Criteria

The criteria of our geo-related tool includes cost, availability, and accuracy. As an open source project, we aim to minimize costs while providing accurate results. As for availability, our application requires that the API supplying geo-related information and data be consistently running, otherwise our ability to determine aerosol content would be drastically reduced. Looking at our accuracy requirement, we need an API that will supply geo-related information not only on time, but accurately. If we receive information that is not accurate, our algorithm will have poor performance results and be incapable of adequately determining the aerosol content of user uploaded images.

D. Selection

After discussing available technologies with our client, we decided that we will be using MISR for gathering geo-related data. It is the only satellite available for public use of the information, and more importantly it also provides highly accurate data. Our client works at NASA JPL, the company which currently commissions the satellite. Therefore, we will have someone on our team with prior knowledge on how to properly analyze and use all of the data. Moreover, this API satisfies all of our criteria mentioned above. The results from MISR are public information, so there is no cost associated with using MISR data. MISR also has no plans of being decommissioned, so the data will consistently be available assuming MISR is not decommissioned. Finally, MISR has 9 angles from which it analyzes Earths atmosphere, so the results and data provided by MISR are highly accurate. Ultimately, it is clear that MISR is the ideal API for our geo-related data needs.

X. Documentation/Development of API Documents

A. Sphinx

When it comes to the documentation and development of our API documents, our team requires the use of an API to create professional, informative, and concise documents. Sphinx is a documentation tool which allows its users to create

both intelligent and aesthetically pleasing documents. This APIs output formats include HTML, LaTeX, and several others. Moreover, it allows its users to perform semantic markups, as well as use automatic links for citations, glossary terms, etc. It also provides a hierarchical structure as well as code handling tools. Overall this API provides a multitude of functions that serve us well when completing our documentation phase.

B. Goals

The goal of using an API for documentation and the development of API documents is to produce documents that will be both professional and informative. As we develop our Aerolyzer application and once it has been completed, we will need to develop both coding and usage documentation for our users. Without documentation on our code and how to use our application, Aerolyzer will be minimally effective as users will be unable to adequately upload images and receive information on their aersol content. Therefore, it is imperative that we use a documentation API which provides the tools and functions necessary to develop clear, concise, and professional documents.

C. Criteria

As for the criteria of our applications documentation and development of API documents, there are several factors that we must take into consideration when choosing which API to use. Such features include price, availability, and tools provided. As an open source project, we need to ensure that the API being used for documentation is affordable. Moreover, we need to ensure that the API is consistently available throughout our documentation phases and produces outputs that are compatible for all users. Finally, we need to make sure that the specific tools and functions of the documentation API we choose provide us with the abilities necessary to create professional and informative documents that will satisfy our documentation requirements.

D. Selection

After conferring with our client, we determined that Sphinx would suffice as our documentation API for several reasons. First of all, the quality of the documents produced by Sphinx will exceed our expectations for code and usage documentation. Moreover, the tools provided for semantic markup, automatic links, and other various functions will aid in our documentation process greatly. Sphinxs ability to output documents in HTML, LaTeX, and various other formats meets our documentation compatibility needs, providing most users with access to our documentation. Overall, Sphinx will be a great addition to our set of technologies used for the Aerolyzer development process.

XI. Conclusion

In conclusion, several technologies will be used to fully implement the Aerolyzer application. Our chosen technologies for application-side functionality include Django for the web interface, MySQL for storage, ExifRead for EXIF data extraction, Dropzone JS for uploading images, and Weather Underground and MISR for weather-related data. For the development team, we will be making use of Travis CI and Coveralls for testing, PyLint for our enforcing a coding style, and Sphinx for documentation. These selections were made based on research into different options and using criteria specific to our needs. A few of our technologies for pieces of the project have already been chosen and put into use; as a result, comparisons were not needed for these cases.[25]

REFERENCES

[1] Pylint. Introduction. [Online]. Available: https://pylint.readthedocs.io/en/latest/intro.html

[2] PyDev. Pylint. [Online]. Available: http://www.pydev.org/manual_adv_pylint.html

[3] J. Castro. (2016) Review of python static analysis tools. [Online]. Available: http://blog.codacy.com/2016/01/08/review-of-python-static-analysis-tools/

[4] Pylint. Tutorial. [Online]. Available: https://pylint.readthedocs.io/en/latest/tutorial.html

[5] A. S. Foundation. (2016) Apache solr - features. [Online]. Available: http://lucene.apache.org/solr/features.html

[6] I. MongoDB. (2016) Mongodb for giant ideas. [Online]. Available: https://www.mongodb.com/

[7] Oracle. Mysql. [Online]. Available: https://www.oracle.com/mysql/index.html

[8] DB-Engines. Mongodb vs. mysql vs. solr comparison. [Online]. Available: http://db-engines.com/en/system/MySQL%3BSolr%3Bmongodb

[9] M. Corporation. (2016) Cve security vulnerability database. [Online]. Available: http://www.cvedetails.com/

[10] Cloudinary. Admin api. [Online]. Available: http://cloudinary.com/documentation/admin_api

[11] P. S. F. Python Package Index. (2015) Exifread 2.1.2. [Online]. Available: https://pypi.python.org/pypi/ExifRead

[12] P. P. Fork). Exiftags module. [Online]. Available: http://pillow.readthedocs.io/en/3.4.x/reference/ExifTags.html

[13] threespot. (2016) Django. [Online]. Available: https://www.djangoproject.com/

[14] A. Ronacher. (2016) Flask. [Online]. Available: flask.pocoo.org/

[15] M. D. Pierro. (2016) Web2py. [Online]. Available: http://www.web2py.com/

[16] J. Koskelainen. (2016) Web frameworks for python. [Online]. Available: https://wiki.python.org/moin/WebFrameworks

[17] T. W. Company. (2016) Weather underground. [Online]. Available: https://www.wunderground.com/weather/api/

[18] T. CI. (2016) Travis ci. [Online]. Available: https://travis-ci.com/

[19] L. H. Industries. (2016) Coveralls. [Online]. Available: https://coveralls.io/

[20] M. Meno. (2016) Dropzonejs. [Online]. Available: http://www.dropzonejs.com/#

[21] Ospry. (2016) Ospry. [Online]. Available: https://www.ospry.io/docs

[22] Cloudinary. (2016) Cloudinary. [Online]. Available: http://cloudinary.com/

[23] D. Hough. Dropzone and cloudinary. [Online]. Available: https://danhough.com/blog/dropzone-cloudinary/

[24] N. JPL. (2016) Misr. [Online]. Available: misr.jpl.nasa.gov/Mission/

[25] Sphinx. (2016) Sphinx. [Online]. Available: http://www.sphinx-doc.org/en/1.4.8/index.html

IV.2 Tech Review Changes

We changed one technology change throughout the course of the year. The change we made was using SOLR instead of MySQL for our database. We decided to make this change after the scope of our project changed and SOLR fit our database criteria better than MySQL did. SOLR had much better scalability compared to MySQL. SOLR is also completely open source and is part of the Apache brand.

V Weekly Blog Posts

V.1 Fall Term

V.1.1 Week 1

V.1.1.1 Reilly: Next week, we will meet with Vee on Monday for the first time. Our expectations for the project right now are just to do some more research on atmospheric optics and image/feature detection. I will also do more research on UI for our app and check out some of similar apps that Kim shared with us.

Last Friday was our first meeting with our client, Kim Whitehall and Lewis McGibbney from NASA JPL. After our meeting, Kim sent us some resources via links to look through to help us get started I looked through these and got a better understanding of our project, which was really helpful for working on the problem statement document. Based on my interest in UX, I was given the tentative team role of looking into the UX component of our app; I explored some options for us to display our photos, as well as how the app will look/feel, by researching other photo sharing and feature detection apps (both for Android and iOS).

Over this past week, we completed our problems statement assignment by creating the LaTeX file and Makefile to compile a PDF for Kim and Lewis. They gave us feedback on our draft and approved the final version, which was signed and sent back to us to turn in.

Our Github repo was also created this week; I pushed all of our problem statement files into a new directory.

As a group, we aren't sure whether we are able to make the Github repo public and use an organization or if we are restricted to just private repos. Kim and Lewis are going to contact the professors about this. Personally, I haven't encountered any problems this week.

V.1.1.2 Sophia: This week, we will be continually doing research on our assigned topics. I am working on getting acquainted with Rust. We will be meeting with our TA on Monday to check in on our project progress.

On 10/14 we met with out client Kim Whitehall and Lewis McGibbney. On 10/9 Dr. Whitehall sent us numerous resources to research during the week. I did a lot of research on the Aerosol Content and Rust. This week we also wrote and all signed our problem statement. The problem statement summarized what us and our clients expectations were for this senior capstone project. All of the parties signed the statement. We also set up this github page and will started updating the WIKI page.

Our group and our clients have been working really well together. The only problem I have encountered so far is that the papers regarding the Aerosol content have been difficult to understand, but after some research I am confident that I will have a firm understanding on the topic. Another problem is having a private repo but having a public repository is causing some issue in Github.

V.1.1.3 Jesse: For this week we will continue to conduct individual intensive research to gain background knowledge on the topics at hand. We will be meeting with our TA this Monday to analyze our progress.

We met with our client last week on Friday to determine what the goals of our project will be. We identified individual research agendas, and read articles that helped us to build knowledge on the subject at hand.
So far our group has worked well together, the only troubles we have encountered have been difficulty limiting our project to certain main goals, as well as researching articles on the topic at hand. Our weekly meetings have been well put together, but we hope to work on having productive conversations throughout the week as well.

V.1.2 Week 2

V.1.2.1 Reilly: This upcoming week, our team plans on continuing research as documents come in from the group mailing list. We will also complete any outstanding tasks that Kim asks us to work on throughout the week, such as contributing more to our readme. We have revised our problem statement and will make further revisions using feedback from our professors, then send it to our clients for more feedback and signatures.
Our requirements document is due (signed) on Friday, so we will be working on that this week with our clients as well.
We met with our TA Vee this week and discussed what to anticipate for our project. He went over some of the expectations for the class and gave us advice on working as a team, interacting with our client, and completing work for the capstone class.
After meeting with Kim on Friday, I setup our updates wiki page and created links for our weekly updates. I have continued working on UX research, as well as reviewing various documents that Kim has been sending to our mailing list. Specifically, I looked at Android UX examples for taking and displaying photos; I also skimmed through the Android design guidelines to get a better feel for typical photo-centric app layouts and formatting.
We resolved an issue from last week by moving our repo under our new Aerolyzer organization and making it public. There are no other problems we've run into since last week.
V.1.2.2 Sophia: This week I focused on reading through the provided research papers more. I have started going through the REST API's Documentation and been watching more tutorials.
This week I plan to do a mini project with REST, and continue to familiarize myself with the research papers. I want to have a solid understanding on the science behind our project by the end of the week. I also need to work on the requirement document and have it to our clients this week.
Our problem statement had to be redone, so we are working on it. We resolved the issue with the private vs. public github repositories. Our github repository is now public, and so are all of our profiles on it.
V.1.2.3 Jesse: This past week our main focus was to conduct more research and further develop our knowledge on the topic of our project. I looked up professors at OSU with a background in optics, and found a professor within the physics department that I plan to meet with to discuss ideas for our project. This week was extremely busy for me, so I did not have as much time to read articles and other papers as I did last week. However, I downloaded iPython which will allow me to play around with MatPlotLib this upcoming week.
This week I plan to become familiar with MatPlotLib, as well as identify some sunset images that we can begin to look at for our project. I also look forward to reading more documents which have been provided by our client. Our team will also be focused on revising our Problem Statement and developing our Requirements Document which are currently due 10/26/16 and 10/28/16 respectively.
One issue we had was satisfying our client's desire for a public github repo, as opposed to having a private repo which our professor specifically asked for. However, we were able to remedy this with permission from our professor to

create a public repo that satisfies both parties needs. Technically speaking, another problem we encountered was having to revise our Problem Statement.

V.1.3 Week 3

V.1.3.1 Reilly: This upcoming week, our team is going to complete the requirements document and have it signed by the client. We will have to complete this by Friday. The team is planning on going over the specifics of our requirements doc on Monday during our meeting with our TA.

My other plans for this week are to continue looking at research documents sent by our client, as well as review the github workflow doc.

This week, the team and I updated our problem statement using feedback from our professors, as well as suggestions from our client. We definitely have a much better understanding of our project now; everyone on the team is on the same page. A final hard copy was turned in, and I committed the final unsigned version on our repo.

We also got started on our requirements document this week. This was challenging at first, but ultimately it's been helpful to figure out what exactly we are going to be doing for the rest of our project. Going through requirements during our meeting with Kim on Friday was helpful for completing the doc as well.

We have not encountered any major problems this week. At first, we weren't sure where to get started with our requirements doc, but after looking through examples and meeting with our clients we have a solid understanding. Our requirements doc is well underway, with a draft complete and final (signed) version on track to be finished by the Friday due date.

V.1.3.2 Sophia: This past week we again focused on reading the research papers provided by our client. I added a description to the Github page, and we finished and signed the problem statement.

This upcoming week were are going to ad coding task. We are also continuing to read research papers, and finish editing and adding to our requirements document.

Our problem statement was returned to us, and we edited and updated it.

V.1.3.3 Jesse: This past week we again focused on researching while also accomplishing a few tasks. I read a couple of papers supplied by our client, as well as added the Apache License v2.0 to our source code in GitHub.

This upcoming week were are going to play around with some code focused on gathering landscape and sunset images from a website. We are also continuing our research while wrapping up our requirements document.

We didn't encounter any significant problems this past week.

V.1.4 Week 4

V.1.4.1 Reilly: This upcoming week, I am going to work on the second coding task given to us by our client. This task involves creating a script in Python that will scrape images and metadata from Weather Underground, and we will eventually use these images in our color-analyzing algorithm.

I will also work on our technical review document this week, as it is due Friday, though the assignment itself hasn't been released yet.

This week, we finished our requirements document as a team. There are some sections that we couldn't fully complete, as they depend on completing further research regarding the color-analyzing aerosol algorithm, but we just made note of that in the document and will add to it as needed. During our meeting on Friday, Kim and Lewis helped us walk through the requirements to make sure everyone was on the same page. It sounds like our clients are very happy with

our requirements and can see the project coming together. We have a pretty good understanding of what we need to accomplish in the coming months and what we still need to find out through exploration and discussion.
I also worked on our first coding task this week. This task involved writing a Python program that used the Weather Underground API to retrieve data about given cities around the world. We were assigned this coding task as practice for retrieving meteorological data needed for the color-analyzing aerosol algorithm. The Wunderground API seems pretty useful for our needs, especially considering it's the free version.
I had some questions about our problem statement feedback, so I talked with Kirsten this week to better clarify what problem we are trying to solve with Aerolyzer. She helped me work through finding the best way to convey our project's goal to others and helped me understand how we could make our web app marketable.
Our team had some problems working on the Gantt chart, as we weren't given any direction on where to start in class. Kim and Lewis helped us write out tasks to be completed until Expo based on our functional requirements, and we converted those into a working Gantt chart.

V.1.4.2 Sophia: This week I will start working second coding task given to us by our client. It is a script to provide a multipanel look at the color channels in a given image. I will also work on our technical review document this week, as it is due Friday.
This week, we finished our requirements document. There are some use cases and requirements we will continue to add as needed while we finish the planning stage of our application. We all agree that it is very exciting to see our project form from ideas, to an actual application.
We had some issues with creating the Gantt chart. With some help from Dr. Whitehall we planned it during our meeting on Friday and we implemented it.

V.1.4.3 Jesse: For this week, we spent the majority of our time finishing up our requirements document. The Gantt chart took us quite a bit of time to complete, mainly because we were unsure how to create one. However, we got help from our client during our weekly meeting and were able to complete it based on that information.
Our problems for this week came from difficulties in creating the Gantt chart. However, as I mentioned we were able to figure it out by communicating with our client.
This upcoming week we have a few tasks to complete. We need to begin working on our tech document, as well as work on a few of the coding tasks that have been assigned to us by our client.

V.1.5 Week 5

V.1.5.1 Reilly: This upcoming week, our team is going to turn in the tech review document. We are also going to work on implementing some of the discussed technologies in our Tech Review. My job will be to implement Pylint for use in our coding environment - to do this, I will setup a configuration file and add a supporting wiki for documentation. I have already done research on Pylint for our tech review, so I should be able to quickly add the config file and wiki. The team is going to discuss these added technologies during our meeting on Friday.
This week, we turned in our requirements document and started the tech review doc. We met with our client on Wednesday to discuss the technologies we would be using, and we came up with 9 different pieces of the Aerolyzer application that we could research to find alternative technologies. My three sections of the requirements doc are the coding environment, image and metadata storage, and extracting EXIF metadata.
I also created an issue for, committed, and submitted a pull request for our final (unsigned) requirements files on our repo.

Finally, we received feedback about our requirements document on Friday. We will need to make the suggested changes to our doc for the final document before we put everything together at the end of the year. We will also apply the general critiques to our documents going forward (e.g. formatting).

Our client gave us several suggestions for improvement on our requirements document and Gantt chart the day that it was due, so we spoke with our TA and were able to turn it in a day later than discussed.

Since we had already decided on a few of the technologies we would be using as a team, our client asked the professor about what was required of the tech review assignment. She didn't think we should spend time searching for alternatives when we had already decided to use and begun moving forward with specific technologies. She told us that he said we only had to compare technologies where possible, meaning we did not have to research alternatives for technologies we had already selected.

V.1.5.2 Sophia: This upcoming week, we are working on and turning in the tech review document. We are also going to work on implementing some of of the technologies into our github repo. Specifically for me I am working on integrating Travis CI and Coveralls support.

We complete and submitted our requirements document and started working on the tech review doc. We met with our Dr. Whitehall on Wednesday to discuss the tech review document, and what technologies we would be using for it. We came up with 9 section of the Aerolyzer application to research alternate technologies, and we each are in charge of 3 sections. My three sections are the web interface, testing, and getting meteorological data.

We received feedback on our requirements document and had some minor issues, so we will be working on fixing the requirements document based on what our TA said. We also had some issues getting our Gantt Chart and requirements done on time, so we received an extension on it.

V.1.5.3 Jesse: We made quite a bit of progress this week, finishing up our requirements document and beginning our tech document. Since there was no school Friday, we met with our client on Wednesday and went over how we are going to complete our tech document. I was assigned sections including Upload Photos, Getting Geo-related Data, and Documentation/Development of API Documents. I also looked into one of our coding assignments from our client, and look forward to working on it further.

We had no major problems, however our client wanted to email our professor about the necessity of the tech document. She was concerned because we already have most of our technologies figured out, so she wanted to make sure we weren't doing unnecessary work.

For this upcoming week, we will finish up our tech document and turn that in. We also have individual coding assignments that we will be working on throughout the week.

V.1.6 Week 6

V.1.6.1 Reilly: This upcoming week, our team is going to begin working on our design doc. We will look into the IEEE Std 1016-2009 document and research examples to get started with our design. We will also talk to our client about the design doc as we work on it (via slack) in order to clarify anything we aren't sure about. Finally, we will need to discuss which parts of the design doc each of us will work on in addition to writing about our specific technologies from the tech review. The design is our only class assignment to work on this week.

During our meeting Friday, we went over some of the code guidelines that should be implemented in the Pylint file. I am going to continue working with the team to come up with a style for Aerolyzer. Should they change as time goes on, I will add these configurations to the aerolyzer_lint_file in the repo.

On Monday, I turned in our tech review files via github. For the tech review, I researched technologies for our coding environment, database, and EXIF metadata extraction. Our team had already chosen to go with Pylint for the coding environment, so I talked about moving forward with that in the tech review. Then, after completing research and analyzing tradeoffs, I concluded that we would use MySQL for a database and the ExifRead Python package for extracting EXIF data. I also started our design doc by creating a tex template, along with other needed files (such as a Makefile, IEEE files, and a bib file). We aren't yet sure what sections will be required for the design, so at the moment our template has the same sections as the IEEE Std document. We will change these once we get a chance to discuss the exact format with our TA and/or professors.

Lastly, I committed a Pylint file for our project called aerolyzer_lint_file in the repo. I will be in charge of maintaining this file going forward. I also added our Pylint information to the wiki. Our TA told me I could install files on the school server without root privileges using the command "python setup.py install –user," and this allowed me to get Pylint setup in my own coding environment.

I ran into a problem checking in our tech review files: somehow I was unable to push changes to my specific branch for the tech review issue I created. I ended up adding files manually via the online github interface in order to meet the midnight deadline. Over the next few days I tried to figure out what went wrong, but I'm still not sure; I was able to get back on master and then on a new branch to check in my pylint files with no issues. The problem seems to be fixed, though, so hopefully I don't run into this issue again.

V.1.6.2 Sophia: This week, our team will begin working on the design doc. Since our client will be out of town for the next few weeks, we will be communicating via slack. I will also work on adding more content to the .travis.yml file I added to the code base last week.

We turned in the tech doc on Monday. I was in charge of 3 sections, and finished my research and we turned in the document on time. I also worked on implementing TravisCI and Coveralls to our code base. This took awhile because I have never used TravisCI or Coveralls before, so I had to do some research to figure out how to implement them.

I ran into the problem of implementing my .travis.yml file to integrate TravisCI to our code base. I was confused about what to add to the content of it, but Lewis was really helpful during our team meeting and gave me an example of what to add to it. I also ran into the issue of adding Coveralls to the main repo, and not just my personal repo. I learned that I have to go into the settings for the main repo and change the settings to allow coveralls to be implemented to it.

V.1.6.3 Jesse: This week we finished up our tech document and submitted it to github. We each completed 3 of 9 sections for the tech document, and after doing some research on certain technologies we were able to figure out what tools we will be using for our project. I looked into a personal assignment given to me by our client which requires me to create a skeleton for Sphinx documentation.

Overall we did not really encounter any problems this week. We had a few issues committing our changes to our repository, however I believe we figured them out (I was not the one trying to commit but I see the changes so I assume they were fixed). I was also slightly concerned that our tech document may have not been received due to these issues, but hopefully that is not the case.

For this upcoming week, we will not be meeting as a group due to the Thanksgiving holiday. However, we will be communicating through email and Slack. I will also be working on creating the actual skeleton for the Sphinx documentation, as well as offering help to Sophia on getting our framework setup in github.

V.1.7 Week 7

V.1.7.1 Reilly: This week, I am going to continue working on the design document and the progress report. We will (hopefully) have the design document finished by Wednesday, then receive feedback and make changes before getting signatures and turning in a hard copy on Friday morning. Throughout the week I will also continue adding to the progress report based on how we decide to divide up the work as a team. We will not have a meeting this Friday due to our client being on vacation.

This past week, I have worked on the design document and completed close to half of it. I added a tex template with filled in sections to our github repo. I also started a document for our progress report. Finally, my pylint code pull request was approved and merged.

We are confused about our feedback on the tech review. Our client told us that one of our professors said we did not have to compare different technologies for pieces of the system that we had already moved forward with implementing, as that would be an inefficient use of our time. However, a different professor graded our assignment, so we believe there was a miscommunication and we received a poorer grade as a result.

V.1.7.2 Sophia: This week, I will work both the design document and the progress report. We will try to finish everything by Wednesday to send to our clients for feedback. I will also work on fixing the TravisCI yml file by changing the content based on feedback from our client.

This past week, I worked on integrating travisCI with our code base.

I implemented the travisCI yml file, but there was still some issues with the content so I need to go back and fix/add some things to it. I also was confused about the feedback on the tech review, so I will go in and talk to our professors about it.

V.1.7.3 Jesse: This week, we mainly worked on developing the design document and looked towards the final progress report. Since we need to run our design document by our client for approval, our plan is to complete the document by this upcoming Wednesday.

This week we did not really encounter any problems. Aside from issues in the grading on our tech review, and not knowing our score for the requirements doc, we did not have any issues.

For the upcoming week, we plan to finish our design document which is due Friday, and start on our progress report. Our goal is to complete the design document by this upcoming Wednesday so that we may get approval by our client before the submission deadline.

V.2 Winter Term

V.2.1 Week 1

V.2.1.1 Reilly: This week, we met with our client and discussed our plan for the upcoming term. Using our Gantt chart from last term as a guide, the team will begin image scraping and designing our web interface this week. Though we will work together to accomplish these tasks, I will specifically focus on image scraping while Sophia and Jesse will focus on the wireframe prototype for our app.

Over winter break, I updated our tech review and design to reflect our choice of Solr for our data storage rather than MySQL. Our client mentioned that scalability will be our most important database criteria, and adding this criteria revealed that Solr would be a better option than MySQL or MongoDB. We will therefore implement our database with Solr going forward.

We did not encounter any problems this week.

V.2.1.2 Sophia: We are following the out Gantt chart so we will be starting to work on the web interface this week. Jesse and I will focus on making a wireframe prototype using Balsamiq. Reilly will work on image scraping Weather Underground.
We did not encounter any problems this week.
V.2.1.3 Jesse: After meeting with our client, my plans for the coming week are to work with Sophia to create a wireframe of our website.
Since the past week was Winter Break, I did not really make any progress since last week.
I did not encounter any problems.

V.2.2 Week 2

V.2.2.1 Reilly: This week, we will be updating the wireframe prototype and figuring out which Bootstrap theme we will be using. Our client suggested some changes for the website design; we will implement these in our wireframe and complete this task by our meeting next Friday.
I pushed our updated Tech Review and Design to the repo. I also pushed the script for image scraping along with the image metadata as JSON files to the repo, then submitted a pull request.
The images themselves are in our shared Google Drive - our client will use these images for initial testing. The Python script uses the Beautiful Soup Python library to scrape the Weather Underground image gallery for pictures. Included in the retrieved data is the date taken, image filename, image URL, comma-separated categories for the image given by Weather Underground, and URL for the gallery page. This metadata is stored in a list of namedtuples and converted to JSON before being written to an output file. All images are then downloaded to the current directory. A command-line argument can be given to change the Weather Underground gallery page used to retrieve images.
Our client mentioned that uploaded images will need size and resolution restrictions. We will need to add these requirements to our Requirements doc at some point.
V.2.2.2 Sophia: This week we will be taking feedback from our client and working on changing the wireframe prototype and figuring out which Bootstrap theme to use.
I updated our travisCI.yml file. There are still some things I need to add. I also helped design our wireframe prototype. Some changes are still needed before our final wireframe is done.
Our client noted a lot of changes in our wireframes, so we will need to change some things. She also told me to add some things to the travisCI file which I am still a little confused about, so I will need to research more into it.
V.2.2.3 Jesse: For this upcoming week, my plans are to update the wireframes so that they are much lower-level and similar to that of a prototype, as requested by our client.
This past week, Sophia and I used Balsamiq Mockups to create wireframes for our website. We tried to keep it at a high level and not get bogged down in the details.
One problem I encountered this week was miscommunication between our client and Sophia and I. Sophia and I both assumed wireframes to be more high level, while our client was expecting the wireframes to be more in-depth and similar to a prototype.

V.2.3 Week 3

V.2.3.1 Reilly: This week, we are going to narrow down a specific Bootstrap theme and create our site's Home and About pages. This will give us CSS and a general skeleton to work with for implementing the rest of the site.

We will also be gathering EXIF data from various mobile images and aggregating info about mobile-specific tags. We will use the results to refine our restrictions, specifically the requirements that the image be taken on a mobile phone, free of editing/filtering, of type .jpg, and greater than a minimum resolution. We will be using Apache Zeppelin to accomplish this.

Finally, we will be looking into Exif.js as a way to import EXIF data from user's images in Python/javascript.

This past week, we worked on updating the wireframe prototype and created a working wireframe to guide our implementation. We also talked about the database design and came up with the tables for our schema. Our team is on the same page for our data flow, our UX, and how we will get data from users to our server.

Since we worked on our database design and made some changes, these will eventually need to be added to our Design doc. Additionally, we will probably end up using Exif.js for our implementation (rather than the ExifRead Python library); this change will need to be reflected in our Tech Review and Design doc as well. Finally, we will need to update our requirements to include a constraint for uploading only .jpg images.

V.2.3.2 Sophia: This week we will continue on working on the wire frame using Balsamiq. We will work on refining the UX/UI to make sure our users have no confusion while using our application. We will also pick a Bootstrap theme to use throughout our web application. This will give us a skeleton to create our application on.

We will also be gathering and analyzing EXIF data extracted from various images taken on various mobile phones. We will be using this information to finalize our restrictions we will place on the images users can upload. We will specifically be using this to refine the size of the image we allow users to upload, but will also gather other useful data.

This past week we worked on creating a wireframe using Balsamiq. We also designed and made a UML diagram that represented our database schema.

We had some corrections to make to our UML diagram after meeting with our client.

V.2.3.3 Jesse: For this upcoming week, my plans are to look into Python's exifread, and see how the EXIF data of a mobile image can be extracted.

This past week, we updated and completed the Balsamiq mockups for our website, which are now much lower-level and similar to that of a prototype.

I did not encounter any problems this week.

V.2.4 Week 4

V.2.4.1 Reilly: This week, we will be working on implementing a Django framework for the frontend of our website and figuring out how to integrate our current HTML/Bootstrap CSS within that framework. I will also push our current HTML/CSS for our home page mockup to the Aerolyzer_Website repo.

Last week, we picked a Bootstrap theme and created our site's home page. We also gathered EXIF data from various mobile images and aggregated info about mobile-specific tags. Those results enabled us to refine our restrictions, and we now specific EXIF tags/other values to check for when creating our script to verify user images.

We ran into some issues trying to setup a Django framework with the Bootstrap HTML/CSS that we already implemented. We will have to work through some tutorials or use other resources this week in order to get that set up.

Also, we will need to update our Requirements to reflect our updated restrictions for uploading a photo. We now have determined 8 different requirements that need to be met in order for our color-analyzing algorithm to work correctly.

Finally, we decided to use ExifTool instead of ExifRead to gather EXIF metadata and will need to update our Tech Review accordingly.

V.2.4.2 Sophia: This week we will be working on downloading and going through the Django tutorial. Jesse and I will also start writing the pseudo code for the restrictions for the user to upload images.
Last week we picked a Bootstrap theme for our web application. We also gathered EXIF data from various mobile images using ExifTool. Reilly uploaded the images to Zeppelin because both Jesse and I could not get Zeppelin to download properly on our computers. We now have the information we need to finalize the restrictions we will place on the images to upload.
I could not get Zeppelin to download properly on my computer. I also had some issues to setting up Django on my computer, but after some troubleshooting it worked. Using Bootstrap with Django has been a little confusing, we will be going through some tutorials to grasp how to use Django.
V.2.4.3 Jesse: For this coming week, I have been assigned to work on the backend python code which will process the image's EXIF data and verify that it meets our restrictions. Therefore, I will be working on writing pseudocode for that.
This past week, I played around with python's exifread, and got a decent understanding of how we can extract the EXIF data from a mobile image.
The only problem I encountered this week was difficulty downloading a program recommended by our client. However, this was resolved.

V.2.5 Week 5

V.2.5.1 Reilly: This week, I will continue working on implementing our website's frontend using Django. Specifically, I will change our repo and directory name in Django to "apps", push the current code, and try to add login/sign up functionality. There are also a few UI changes suggested by our client that I will implement.
Additionally, we will finish updating our documents and add them to the OneNote notebook, as well as our midterm progress report/slides/video.
Last week, I moved our HTML/CSS code to our Django framework and updated the mockup website with more functionality. Sophia created our webapp repo that I pushed our code to. I was able to figure out how to integrate our bootstrap implementation to Django.
I also created our team's OneNote notebook and setup the structure, adding our working documents and current revisions.
No problems this week.
V.2.5.2 Sophia: I will download and research how to use SOLR. I will start looking into how to make user accounts and work on login functionality.
Last week Jesse and I worked on writing the psuedocode for the Python Scripts, I also went through the Django Tutorial.
No problems this week.
V.2.5.3 Jesse: This upcoming week, I need to revise the pseudocode according to our clients recommendations, convert it to python instead of pdf format, and complete the progress report and presentation/slides with my team.
This past week, I created the pseudocode for the image restrictions with the help of Sophia and our client.
I did not encounter any problems this week.

V.2.6 Week 6

V.2.6.1 Reilly: This week, I will work with Sophia to integrate Solr with Django and implement the login/signup functionality. My plan is then to start working on some of the EXIF code, specifically figuring out how to execute

Python scripts in Django and then testing a script to run ExifRead on images.

This past week, I continued working on implementing our website's frontend using Django. I changed our repo and directory name in Django to "apps", as well as changed the Aerolyzer_Website repo to Aerolyzer_App and closed the associated issue. I then pushed the updated code, which included a few UI changes suggested by our client and other fixes. Lastly for the frontend development, I added a signup page and a user dropdown menu for logged in users. Additionally, I updated and shared our OneNote notebook, as well as helped finish and turn in our midterm progress report/slides/video.

We only had one week to work on the midterm progress report, slides, and video, as the assignment was published later than we were told. In addition to updating all documents and creating our OneNote notebook, we did not have much time to develop this week. It was difficult to work on all of these tasks, plus our tasks for the app, in the midst of other midterms during week 5 and 6.

V.2.6.2 Sophia: This week Reilly and I will continue to work on integrating our backend SOLR with fronted Django to implement the login/signup functionality.

I downloaded SOLR and went through the tutorial, I am still working on learning how to create user accounts and integrate SOLR with Django using Haystack. We also completed the progress report slides, video, and paper.

It was a time crunch to get all of the assignments in, next time I wish we had more than a week to get everything done.

V.2.6.3 Jesse: For the upcoming week, I need to revise the pseudocode.py file for image restrictions until it is complete.

This past week, I continued to revise the pseudocode according to our client's recommendations. I also helped complete our progress report and presentation/slides.

The only problem I encountered this week was miscommunication with our client on how she wanted the pseudocode to be created.

V.2.7 Week 7

V.2.7.1 Reilly: This week, Sophia and I will continue implementing the user features and hopefully move on to the gallery and upload photo pages. We decided to use a SQL database for user info and the Solr database for image info. Our idea right now is to have usernames and an ID act as keys for pulling a specific user's pictures into view for the gallery page, so we will try to get that set up this week. Next, my plan is then to start working on some of the EXIF code with Jesse. Specifically, figuring out how to execute Python scripts in Django and then testing a script to run ExifRead on images.

This week, I worked with Sophia to integrate Solr and Django to implement the login/signup functionality. So far, the signup and login works - however, we still need to make sure users are being added correctly. I also added some skeleton user pages, and we turned in our required signed documents.

We ran into trouble trying to integrate Solr and Django, but our client helped us figure it out. I was also concerned about pushing my code and how everything will work in production, so we will see how that pans out in the next couple of weeks.

At some point, we will need to further update our documents by reviewing our client's comments from our midterm progress report. We will also need to add to our database schema in the design doc to reflect our use of SQL for users and Solr for images, as well as our decision to only gather an email, username, and password for the user table.

V.2.7.2 Sophia: This week Reilly and I will start learning how to use PostgreSQL with Django and SOLR. We decided to move on from using SOLR to Postgresql since Django has built in user accounts. We will be using SOLR for all of the images, EXIF Data, Weather Underground information, and aerosol data.
This week Reilly and I worked on using SOLR to get the user accounts, but we had to get the clients help with how to use Haystack.
We ran into trouble trying to integrate Solr and Django but our client helped us figure it out. We are having trouble getting SOLR up and running with django because there is not any good tutorials and resources out on the internet. We are probably going to use Postgres to make the user accounts.
V.2.7.3 Jesse: This upcoming week I need to use the pseudocode.py file to create python classes for the image restriction, as well as a configuration file.
This past week, I finished updating the pseudocode.py file for our image restrictions, as well as merged the Sphinx documentation to GitHub.
The only problem I had this week was continuing to learn python ”tricks” that would improve my code.

V.2.8 Week 8

V.2.8.1 Reilly: This week, we will move on to the gallery and upload photo pages using Solr. The usernames of logged-in users and an ID will act as keys for pulling a specific user's pictures into view for the gallery page. Next, my plan is to start working on some of the EXIF code. Specifically, figuring out how to upload images and pull data (with DropzoneJS and ExifRead, respectively), then running the image verification code.
This week, Sophia and I finished implementing the user features (signup, login, and logout) using the Postgres database. We also added some UI fixes.
Since we switched from using Solr to Postgres for user account storage, it took us more time than we planned to get user functionality implemented. However, we now have a clear path on what to work on this week and think we are still on track.
V.2.8.2 Sophia: This week, we will finish the log-out functionality and start working on using solr for the gallery and upload photo pages. Next, I will start implementing DropzoneJS for drag and drop functionality to upload photos.
This week Reilly and I learned how to use PostgreSQL with Django and SOLR. We decided to use Postgresql since Django has built in user accounts. We set up the user account table, and we can not log in and sign up. We are still working on log-out functionality.
We are a little behind since we switched what database we were using, but we will be able to get back on track after a little work.
V.2.8.3 Jesse: This coming week, I need to convert the configuration file to a YAML file, as well as revise the image restriction functions according to our client's recommendations.
This past week, I used the pseudocode for image restriction functions as a template to develop python classes for image restrictions, along with a configuration file in python.
One problem I encountered was learning how python classes work.

V.2.9 Week 9

V.2.9.1 Reilly: This week, I will work on further implementing the gallery and upload photo functionality. My focus will be on the upload photo page. This work will involve incorporating Solr with html and the Django views for our app.

Finally, I will work on my progress report and the team will work on our poster draft/slides.
This week, I worked on the gallery and upload photo pages using Solr. We got the backend up and are able to search the Gallery and Image tables. I also worked on some UI fixes.
There is definitely a learning curve for using Solr with Django still, as there are not many resources for Solr or Haystack.
V.2.9.2 Sophia: This week, I will finish implementing dropzoneJS. We will also work on the progress report and our poster draft. I will also continue playing around with SOLR to try to get thing implemented before spring break.
This week, we fnished the log-out functionality and started working on using solr for the gallery and upload photo pages.
SOLR and Django are still kind of complicated to use together, but Reilly and I are doing more research.
V.2.9.3 Jesse: For this week, I need to learn how to do assertion testing in python, as well as learn how python's exifread function presents EXIF data for mobile images.
This past week I converted our configuration file to a YAML file as requested by our client. I also revised the image restrictions class according to her recommendations.
The only problem I encountered this week was learning how to create a YAML file.

V.2.10 Week 10

V.2.10.1 Reilly: My plans for this week include writing my individual progress report paper and working with the team to write and record our slides and presentation. I will also try to implement Pysolr within our Django views for the upload photo page, or at least complete more research on how to upload images to our Solr database using Pysolr, Django, and DropzoneJS. I would like to be able to get our upload image backend completed.
This week, I finished up the integration of Solr into Django and my PR was merged. This involved tweaking the Haystack tutorial to implement our own image and gallery models in Django and indexes in Solr, as well as adding an admin-protected search page for us to use. I'll be ending the term in a good spot as far as having all of my code I was working on integrated into the master app branch.
Additionally, I did some research on MISR, Pysolr, and user functionalities in Django (e.g. resetting password, deleting account that we will need to add).
Finally, the team worked on and completed our poster draft.
I will need to discuss the image table schema with our client to make sure the code currently in our repo reflects how we want to store images and other image-related data.
Also, after the bit of research I did on MISR, I think the team will need to talk with our client about how we plan on using MISR datasets to pull geo-related data for uploaded images.
V.2.10.2 Jesse: This upcoming week is finals, therefore my plans for next week are to complete my individual progress report, complete all slides for our presentation, and record our presentation. I will also continue any work on the image restrictions python code if possible, and will mainly focus on updating function names and revising the test suite, as well as creating an exifread class.
This past week I completed some tutorials on assertion testing in python, and created some generic tests. I also played around with exifread to determine how python returns EXIF data for mobile images. Finally, I worked on the poster draft with my group.
One problem I encountered was that I did not know what to test for necessarily in our python code.

V.3 Spring Term

V.3.1 Week 1

V.3.1.1 Reilly: This week, I plan to create pages for our upload photo functionality. These pages will be displayed in-between the upload of an image and the final aerosol results page. I will also create skeleton code for any other pages we will need, in order to have at least existing pages for everything we need in our frontend.

I will also be helping my teammates with connecting different pieces of the project in Django. Since I have been working primarily on the Django framework for our project, I have a good idea of how we will be able to run the needed Python code and display return values.

Finally, we have asked our client for poster draft feedback and will make any necessary changes as that feedback is received.

Over spring break, I did not complete any development. However, I did do further research on MISR and PySolr in order to be prepared for tasks this coming term. I have a good idea of how these two pieces will be integrated moving forward.

No problems have been encountered this week.

V.3.1.2 Sophia: This week, I plan on working on DropzoneJS for uploading images. Drozone will be saving images into files. We will also be working on our poster draft.

Over spring break I did not do any development.

No problems have been encountered this week.

V.3.1.3 Jesse: This upcoming week, I plan to pick up where I left off last term by working on the image restriction functions and creating the EXIF data retrieval class.

This is the first week back since Spring break, and our team did not meet until Friday. Therefore, I did not make any progress since last week.

No problems were encountered this week.

V.3.2 Week 2

V.3.2.1 Reilly: For the upcoming week, I plan to continue working on integrating the pieces of our project together. This will probably involve working with Jesse and Sophia to implement and connect the python code for EXIF data and wunderground. After meeting with our client on Wednesday, I will also work on any tasks she assigns.

This week, I created pages for our upload photo functionality and wrote pseudocode/commented code for piecing everything together. Once all of the python code is written for gathering data (EXIF, MISR, wunderground), the Django views should be pretty well set up for everything to work together. I plan on creating a PR by the end of the weekend.

I also updated our poster draft with feedback from Kirsten and our client. Our team will review and submit the poster draft sometime this weekend.

No problems have been encountered this week.

V.3.2.2 Sophia: For the upcoming week I will be impelmenting Django's file upload instead of DropzoneJS to upload the files.

This last week, I implemented Dropzone JS, but we decided that we would implement a differnt interface for uploading images. We also updated the poster draft.

No problems have been encountered this week.

V.3.2.3 Jesse: My plans for the upcoming week is continue updating the image restriction functions and start a draft of the EXIF data class.
Since last week, I was able to make some updates to the image restriction functions class and explored how to make an EXIF data retrieval class. More specifically, I uniformly shortened the function and variable names, and looked into how to make a class for retrieving the EXIF data.
No significant problems were encountered this week. Moreover, since changing our client meeting to Wednesdays, we did not meet this week.

V.3.3 Week 3

V.3.3.1 Reilly: For the upcoming week, I plan to continue working on integrating the pieces of our project together by addressing our client's comments for our latest PR. This will involve adding a call to exifread, creating a dictionary with GPS tags to send to the wunderground script, and deleting an image if it wasn't able to be verified/processed. I will also work with Sophia to store images in our database once all checks have passed and aerosol data was able to be captured. This will involve using Pysolr, which (from my research) should be pretty easy to implement.
This past week, I completed my PR for upload photo functionality and integrating the pieces of our project together. This involved updating the skeleton pages for uploading photos to have functionality, editing the views and using sessions to piece each part together, and adding comments/pseudocode for how the final product would eventually work (since not all scripts have been written/tested). I also submitted our poster draft 2.
Since we are going to be working on uploading to our Solr database, we will need to review our current schema with the team. I think the current types for exif and weather data (one string) will not be what we want; instead, we will probably want fields for each aspect of the exif and weather data, such as one for GPS coordinates, one for color channels, etc.
V.3.3.2 Sophia: For the upcoming week, I plan to work on having the images save into the database.
This past week, I followed a tutorial and was able to have the images saved into a media foldr.
No problems.
V.3.3.3 Jesse: For the upcoming week, my plan is to alter the EXIF data retrieval class into a more general image data retrieval class which retrieves both EXIF and RGB data. Moreover, I plan to come up with a way to retrieve RGB values, and manipulate them to determine if an image is a landscape/sky.
This week I pushed the changes on the image restriction class from the previous week into a PR, along with a draft of the EXIF data retrieval class.
This week, I realized while creating the EXIF data retrieval class that we might want to make this a more general class, such as an image data retrieval class.

V.3.4 Week 4

V.3.4.1 Reilly: This week, we plan on committing our updated requirements doc and completing our wired assignments. For our project, we will be getting the app ready for expo by completing more features. Finally, we will submit our poster for printing on Sunday.
This past week, I integrated our weather retrieval script and made some changes to the upload photo functionality. I also worked with Sophia to store images in our database once all checks have passed and aerosol data was able to be captured. Overall, I worked on getting our app ready for the code freeze.

We also sent a document to our client for approval regarding updating our requirements.
Lastly, we finished our final poster draft and submitted it to Kevin and Kirsten.
We were unaware of the code freeze on May 1, as the class website has not been updated yet this term. We do not really have enough time to complete all of the functionality we wanted to have May 17 by May 1.

V.3.4.2 Sophia: This week, we plan on updating our requirements doc. We are also going to be doing our wired assignments. For our project, we will be preparing our app to be ready for expo. We also will be submitting our poster to the library for printing.
This past week, I worked on storing images in our database. This will only happen once all checks that Jesse wrote in his scripts have passed. We also finished our final poster draft and submitted it to Kevin and Kirsten.
We are in a time crunch for getting our code ready for the code freeze.

V.3.4.3 Jesse: This upcoming week, we have our code freeze Monday. Therefore, my plans are to solidify the EXIF tags, and debug all of the restriction and image data retrieval functions before Monday. Then I will hopefully build upon the is_landscape function after conferring with our client next Wednesday, and begin making any necessary changes prior to Expo.
This past week, altered the EXIF data retrieval class to be a general image data retrieval class which has functions to retrieve both EXIF and RGB data. Moreover, using OpenCV for the RGB retrieval, I began looking into ways in which we can test whether or not an image is a landscape/sky.
One problem I encountered involved the is_landscape function. I originally brought in the RGB values using matplotlib, but now that our client would like histograms of the RGB values I found it more prudent to use OpenCV. Therefore, I had to rewrite some of that code to produce the histograms using this method.

V.3.5 Week 5

V.3.5.1 Reilly: This upcoming week, I will continue working on the issues in our repos. Mainly, I will work with Jesse to migrate the image restriction code from the App repo to the Aerolyzer repo and update his code in the Aerolyzer repo so that it works within our Django views. This will also involve importing the Aerolyzer package instead of calling functions from files. We will need to return EXIF tags if all checks pass.
This past week, I completed my wired assignment. For the project, I integrated Jesse's EXIF image restriction verification code into our app. I also worked on getting the app ready for expo by moving the weather retrieval script to the Aerolyzer repo and removing it from the App repo, instead importing the Aerolyzer package. Finally, I made issues on both repos to act as guidelines for features/bugs we need to work on for Expo.
I realized we will currently have a few bugs in our app, but after discussing the problems with our client we have created issues on both repos to fix these. They will be remedied before Expo.

V.3.5.2 Sophia: This upcoming week, I will continue to work on updating the requirements document and the wired assignment. I will also clean up the code.
This past week, I completed my wired assignment. I also updated the requirements document.
No problems.

V.3.5.3 Jesse: This upcoming week, I am hopeful that I will get a chance to build upon the is_landscape function before our Aerolyzer code freeze this coming Friday. I will also be working on the WIRED assignment.
This past week, I solidified the EXIF tags and tested the image retrieval functions before our code freeze.
This week I did not encounter any problems.

V.3.6 Week 6

V.3.6.1 Reilly: This week, our team will be getting ready for expo! We won't be working on any more code, other than fixing issues that arise when we push to production. We are also going to turn in our progress report and presentation on Monday. On Friday morning, we will pick up our poster and head to expo.

This past week, I continued working on the issues in our repos. Mainly, I migrated the image restriction code from the App repo to the Aerolyzer repo and update Jesse's code in the Aerolyzer repo so that it works within our Django views. This involved importing the Aerolyzer package instead of calling functions from files. We will need to return EXIF tags if all checks pass. Additionally, I changed our weather API key to use the key associated with our google group's account. I also updated our lint file to make variables camelCase, as well as change our views to copy images to our installDir rather than move them from the media folder. Finally, I worked on our midterm progress report and presentation.

Our team implemented our own code freeze for today, but I still have outstanding PRs that have not been reviewed or merged. Hopefully they will be merged before we push everything to production Sunday.

V.3.6.2 Sophia: This week we are going to be preparing for expo by practicing our elevator pitches. We are also going to finish our progress report and presentation. Progress since last week

This past week I worked on our midterm progress report and presentation. All of my portion of the code has been finished and pushed to the repo.

No problems.

V.3.6.3 Jesse: This upcoming week, I am looking forward to practicing our Expo pitch with our client during our weekly meeting. We are also going to complete our midterm progress report this Monday, as it is due Monday at midnight.

This past week, I completed the WIRED assignment and emailed it to Kirsten. I also looked into the is_landscape function a little, however I did not have much time to complete it.

One problem I encountered this week was not having much time to work on the is_landscape function, as my thesis for physics was due this Friday - the same day as our Aerolyzer code freeze.

V.3.7 Week 7

V.3.7.1 Reilly: This week, our team will meet with our client to reflect and give feedback about our project. We will also complete our three short writing assignments and potentially start working on updating/revising documents.

This past week, we practiced our pitch with our client and fixed a couple bugs in production (one for if an image was too large and another for our Solr database core). Other than that, we really just prepped for expo.

I noticed a few UI bugs during expo that I will make issues for on our repo.

-If you were to redo the project from Fall term, what would you tell yourself?

I would have told myself to start development earlier than winter term. I would also tell myself to revise documents throughout winter term as our design/requirements change.

-What's the biggest skill you've learned?

I've learned how to work on a dev team effectively, using tools like Github and slack to communicate and improve our software/app.

-What skills do you see yourself using in the future?

The team-building and collaboration skills will definitely come in handy in my career. I also improved my Python, HTML, web dev, and UI skills, though I am not sure I will use those in the future (but who knows!).

-What did you like about the project, and what did you not?

I really liked creating a webapp. I hadn't done extensive web development before, and we ended up creating an entire web framework from hosting the site to designing the UI to testing features. What I didn't like was all of the documentation we had to do fall term - though it definitely helped, requiring us to have our requirements completely figured out for us to be graded on before we had even started coding or worked together as a team was not a good programming methodology for us. Completing rough drafts of documents the first half of fall term, starting some development, and being encouraged to revise our docs throughout the project without having to go through a lengthy process of changing them (e.g. formally requesting to change our requirements) would have worked better for our team.

-What did you learn from your teammates?

I learned more about debugging, working as a team to brainstorm, and collaborating to think through software design from my teammates.

-If you were the client for this project, would you be satisfied with the work done?

I think I would be satisfied - we have a working app, issues on the repo for stuff that needs to be fixed going forward, and setup a framework with commented pseudocode for how everything should eventually be implemented/fit together.

-If your project were to be continued next year, what do you think needs to be working on?

This project will be continued next year! The issues in our repo will be what needs to be worked on, including improving some of the data quality and retrieval scripts, as well as UI fixes. Additionally, the atmospheric science code (MISR satellite data retrieval and core algorithm) and displaying aerosol results will need to be worked on.

V.3.7.2 Sophia: If I was to redo the project from Fall term I would start to implement code earlier. We were a little pressed for time in the end, so I would recommend starting fall term instead of winter term. I would also rewrite the requirements document and have the requirements more realistic. We achieved all of our requirements, but we had some really random detailed things like having a back button on the top right spot on the page that were unnecessary. The biggest skill I've learned is working with team members, and allocating tasks within our group. These are all important tasks in the future because software engineering is very team based and being able to work with your team and clients are key to being successful. I'm very excited for this project to be continued next year. I think that we have set up the team next year to be very successful. Phase 2 needs to complete the algorithm, and piece all of the data together.

V.3.7.3 Jesse: I learned a lot from this project, but if I had to choose one thing as the most important thing I learned, it would have to be the importance of developing a design plan and providing thorough documentation. I was initially frustrated by how much documentation we had to do, and that we did not begin coding until Winter term. However, looking back I would say it is definitely going to be useful to the next group to see our thought process throughout the development phase. I was also happy that I got a better understanding of GitHub, and I learned quite a bit about how it provides version control. Conclusively, I enjoyed this project and I am happy that I got to work with our team and client.

VI Project Documentation

VI.1 How our project works

VI.1.1 Project structure

Ou webapp's structure was created using Django, a high-level Python-based web framework. Django integrates our project's frontend and backend, providing functionality with "views" written in Python (see `views.py` in IX) that connect to individual HTML pages. The frontend was designed using Balsamiq and implemented using a Bootstrap theme for the CSS/JavaScript. Our backend consists of two databases: one for user data stored in PostgreSQL and one for image data in stored in Apache Solr.

An API called Haystack is used to provide modular search for Django using Solr, and the PySolr package is used to add image data to our Solr repository. The ExifRead package is used to read EXIF metadata from uploaded images, while WeatherUnderground's Weather API is used to retrieve meteorological and astrological data from the images.

Our GitHub repository for the Aerolyzer webapp contains a `requirements.txt` configuration file listing Python packages required for our project. Other than the above mentioned Django, Haystack, and PySolr requirements, our project is structured to use Gunicorn, WhiteNoise, Psycopg2, and the Django Include Decorator. Additionally, our own Aerolyzer repository within our Aerolyzer organization has been configured as it's own package that can be imported in Python.

Finally, our Aerolyzer codebase uses Pylint to enforce a coding style; our linting file is included in our GitHub repository. Travis CI and Coveralls are used for testing and to verify code quality - testing scripts will be implemented in phase 2, but the infrastructure is in place. Sphinx is also setup to create documentation with a command file present in our repository.

VI.1.2 Theory of operation

As shown below in image 4, the general theory of operation for phase 1 of our web application involves signing up for an account, logging in, uploading a photo, verifying restrictions, retrieving data, and being taken to the aerosol results page. Note: if user enters invalid signup or login account information, error messages are displayed and user is unable to signup/login (respectively).

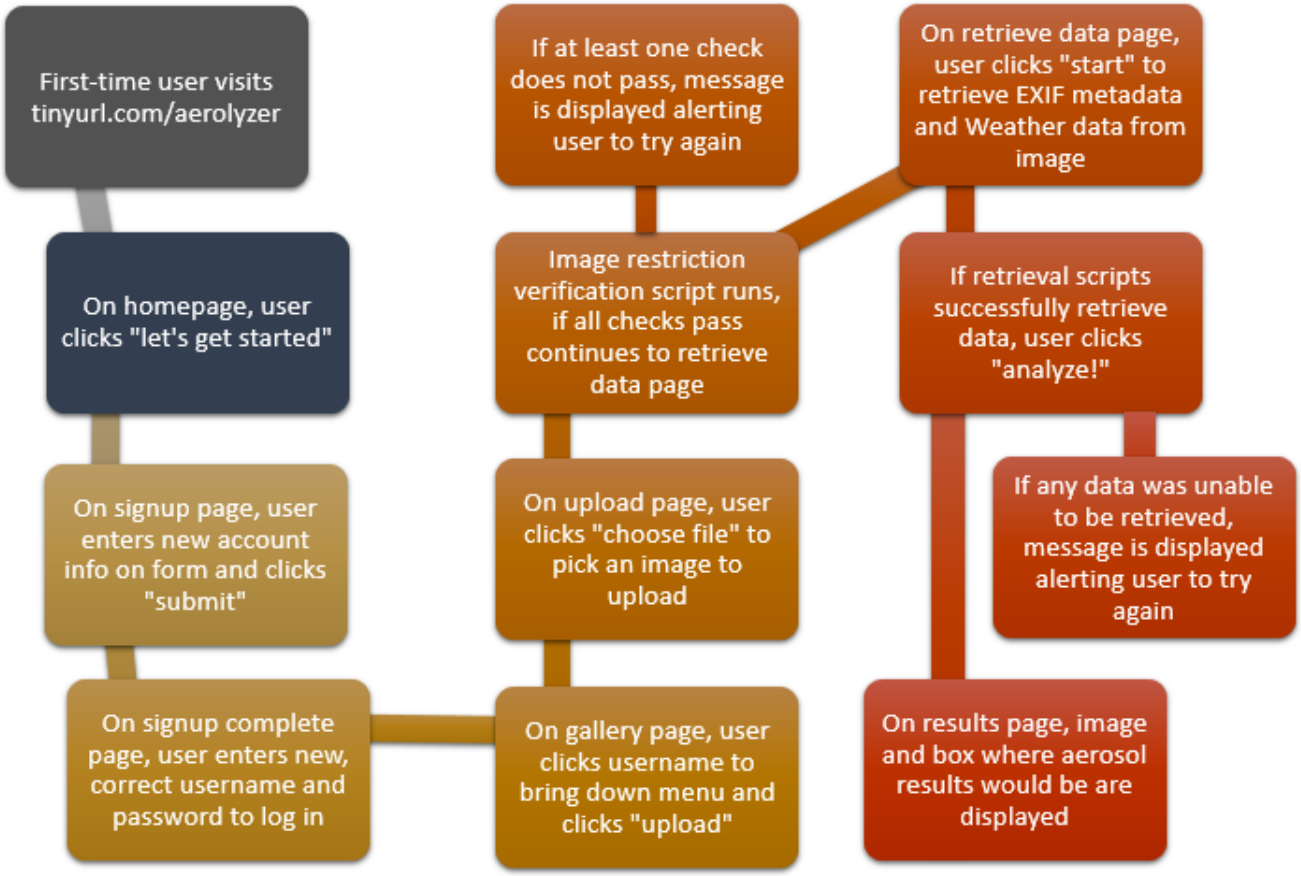


Fig. 4. Theory of operation flow diagram

VI.2 Setup

Because Aerolyzer is a mobile webapp, there is no installation; users simple visit our website at tinyurl.com/aerolyzer in any (relatively) current browser either on mobile or desktop. The site has been tested on the most up-to-date versions of Chrome and Firefox on both mobile and desktop.

To run Aerolyzer yourself (either locally or in production), use our `setup.sh` script, which can be found in our GitHub Aerolyzer_App repository. The script will ask whether you'd like to run the webapp locally or in production. This script has been tested in both Ubuntu and Mac environments.

VII Learning New Technology

Throughout this project, we utilized a wide variety of resources to learn new technologies and broaden our coding knowledge. As each of us were working with technologies for which we were unfamiliar, we collectively had to learn how to find answers to our questions using the internet and various other resources. Our primary resource was undoubtedly our client, Dr. Kim Whitehall. As a result of her extensive knowledge of the technologies used in our project, she was consistently able to provide us with help in learning these technologies, whether through direct teaching or simply pointing us in the direction of another useful resource. We also made use of a variety of websites to

aid in our learning of these technologies, including specific documentation websites for technologies including Haystack, Django, and Apache Solr, as well as online learning communities for programmers like Stack Overflow and TutorialsPoint. A list of the websites utilized throughout our project can be seen below listed in order of helpfulness.

1) http://www.stackoverflow.com
2) http://www.tutorialspoint.com/
3) http://www.tangowithdjango.com/book/chapters/bootstrap.html
4) http://mayukhsaha.wordpress.com/2013/05/09/simple-login-and-user-registration-application-using-django/
5) http://docs.djangoproject.com/en/1.11/
6) http://django-haystack.readthedocs.io/en/master/
7) http://lucene.apache.org/solr/

VIII What was Learned

VIII.1 Reilly

VIII.1.1 Technical information

The technical information I learned was probably my biggest takeaway from this senior capstone project. I had not done extensive web development before (beyond simple HTML and CSS), and we ended up creating an entire web framework: hosting the site, designing the UI, testing features, and more. Specific technical knowledge I gained includes web development (HTML, JavaScript, Bootstrap), Python/various available Python packages, Git, working with PostgreSQL and Apache Solr database, and using the Django framework.
Furthermore, our client explained various parts of our project that she completed as a team member and project manager, such as how to create our own Python package, set up an AWS box, and write installation scripts. Although I did not personally implement these aspects of our webapp, I still learned a lot of technical information about them from our client.

VIII.1.2 Non-technical information

Through this project, I learned how to collaborate with other software developers from various backgrounds to create, modify, and test the same code on a public site such as GitHub. Our client was extremely helpful in teaching us best practices for version control, code styling, and code reviews as a team. I also learned how to create efficient and well-written documentation with a group, both in-class and from our client this includes how to write a Software Requirements Specification, a Software Design Specification, and READMEs.
Additionally, we used various types of technologies for which there were not always good (or any) documentation. This difficulty made me greatly improve my ability to research, wade through tutorials/blogs, and exhaust my resources. Whether debugging or just trying to figure out where to start, my teammates and I were always able to figure something out, and my confidence in my software development capabilities has improved as a result.
Finally, I gained non-technical knowledge regarding several tools for software development. Our client did a great job of introducing us to common software, such as slack, Google Groups, and a GitHub.io team page.

VIII.1.3 Project work

I learned the purpose for designing requirements and specifying your project design before you start working. Had we not completed these documents beforehand, we probably would not have taken the time to get together as a team and thoroughly discuss our vision for the project. Though our requirements and design were certainly not perfect from the get-go, we were able to start with a foundation of what we wanted and work from there as our needs changed. Along those lines, I learned that everything does not always go as planned when creating software and that is okay looking back at our original documents, there were definitely gaps in our planning, and in general we ran into many issues over the course of our project. These experiences only make me more confident that I will know what to think about next time I start a project, as well as be able to push through (as we did this past year) when there appears to be no solution to an issue.

VIII.1.4 Project management

As we managed our project, I learned how to work with individual team member's strengths while also trying to improve our weaknesses and allow for gaining more experience in areas where we were not as skilled. I also improved my understanding about time management in terms of a project timeline: completing a task involves giving yourself time to complete all aspects of that task (research, design, implement, debug, integrate, etc.). A final aspect of project management I learned includes the importance of communication. Making sure every team member was on the same page was extremely helpful throughout our project. This information is especially important when multiple people are creating pieces of code that eventually must work together.

VIII.1.5 Working in teams

Being on a team consisting of my peers and our client taught me how to work with people of different skill levels. My student team members and I are all about on the same level, though our programming skills differ in specific areas based on our experiences. On the other hand, our client's skills are of course well beyond my own; it was great to learn from her and also learn how to work with someone so advanced in programming/development. My student team members taught me a lot also: I have become better at debugging, working as a team to brainstorm, and collaborate to think through software design from different perspectives.

VIII.1.6 If you could do it all over, what would you do differently?

Without considering being able to change the course organization itself, I would have started development earlier than winter term I also would not have spent so much time worrying about specific details of our documentation. In general, it would have been better to make sure our documents covered all aspects of our project rather than try to have everything figured out on a small scale; we would have benefitted from larger-scale thinking in our design and requirements. Overall, though, I am really happy with where our project ended up. Our client was great, my team members were great, and my experience overall was positive.

VIII.2 Sophia

VIII.2.1 Technical information

This project helped me learn so much more about databases, testing, and web frameworks. Coming into this project, I had not taken databases yet so they were not my strong suit. My goal was to finish this year with a stronger

knowledge of databases and I accomplished my goal. One of the things that helped me the most was doing the tech review. This document allowed me to research all of the different types of databases, and learn the pros and cons of each of them. Throughout our project we switched databases, so this allowed me to learn how to use more than one database. Even though we did not actually implement any tests, I set up TravisCI, Coveralls, and a testing suite. By researching each of these technologies, I have learned a lot more about how to test code which is essential to creating a good project. Before this project, I had not used a web framework before. This project has taught me that frameworks are extremely useful and save a lot of time. A great thing about learning Django is that frameworks are pretty similar across the board so learning how to use another framework will be a lot easier.

VIII.2.2 Non-technical information

The non-technical skills that I have gained are documentation, communication, and responsibility. This year I have written more documents than I have in any other class.During the writing process I remember thinking that the work was tedious, but looking back the documentation really helped make the project go smoother. The tech review was a great way to look at all of the available technologies to make sure that we were using the best technology for our problem. I feel like I have grown a lot as a writer and I really appreciate all of the feedback we have gotten from both the professors and the TA's. This class has also helped my communication skills. Between practicing our elevator pitches in class, doing a mock poster presentation with another group, and practicing our pitches with our client, I felt really confident and ready for Expo and I had a lot of fun presenting to people. This class has also helped me with time management and responsibility. Setting deadlines for myself was key to doing my work well and in a timely manner.

VIII.2.3 Project work

My biggest takeaway from project work is that a project is not all about the technical part and coding, a successful project requires planning and documentation. We would not have been as successful if we had start coding day one without doing any of the research and planning that we did. We would have been switching around technologies a lot more, and forgetting to do certain requirements.

VIII.2.4 Project management

A huge part of why our team was so successful is that we played to our strengths and helped cover each other's weaknesses, we delegated every task, and we helped each other. One thing I really liked about this project is that every team member took on tasks that were not our strong suit, but we helped each other through it and that is how to grow as a team and as a software engineer.

VIII.2.5 Working in teams

Through this project I learned that working well in teams helps a person grow so much more than a person working by themselves. Our group did a lot of bonding early fall term and that made a huge difference in our team dynamic the rest of the year. We knew that we could count on each other to do good work, and help one another when we were struggling with some issue.

VIII.2.6 If you could do it all over again, what would you do differently?

If I were to do all of this over again I would work harder and think through the documentation. Having a good plan and preparing makes all of the difference when making an application. Our team did really well in planning the

project, but we could have thought through our design document more. Good planning can really save a lot of time in the long run.

VIII.3 Jesse

VIII.3.1 Technical information

Over the course of this year-long project, I learned a great deal of technical information for which I had no prior experience. Two of my main goals heading into this project were to gain experience in backend development, as well as develop my knowledge of Python programming. Overall, I believe that I accomplished these goals, as I spent the majority of my time working on the backend using Python to handle and process the image data. Through my technical contributions, I would argue that I developed a working knowledge of the Python programming language, as well as gained invaluable experience in backend software development.
Coming into this project, I also had little to no prior experience with version control. Therefore, one of the greatest pieces of technical knowledge I garnered from this experience was how to use GitHub, as well as the importance of maintaining version control throughout a project.

VIII.3.2 Non-technical information

In regards to non-technical information, I learned a great deal about documentation and communication from this project. Through the development of our requirements document, design document, and technology review, my understanding of documentation and its role in project development grew substantially. Moreover, over the course of this year, I also learned a lot about communication and how to work as a team from my team and client.
Through our constant collaboration, my teammates helped me to develop my communication skills, and allowed me to contribute significantly to this project. Moreover, I not only learned how to communicate, but I also learned how to collaborate and utilize the ideas of all team members. Finally, through constant interaction with our client, I learned about the importance of communication and honed my ability to communicate effectively over the course of this project.

VIII.3.3 Project work

While I have learned a great deal about project work throughout this project, one of the most important takeaways for me has been that project work is not all coding. In fact, most project work, especially in the early stages, focuses on documentation and developing a design plan which the project will reference throughout development. However, I also learned that changes to the documentation and design plan are inevitably going to occur. Therefore, being able to adapt and maintain accurate documentation is of the utmost importance.

VIII.3.4 Project management

As far as project management goes, I learned a lot about how to collaboratively delegate work, utilize strengths and weaknesses of various team members, and how to manage time effectively. Throughout our development process, we had several tasks which we delegated out to specific team members in order to efficiently complete work. We also focused on improving our weaknesses while utilizing individual strengths to assist each other throughout the learning process. Through developing collaborative documents and dealing with numerous deadlines, I learned a lot about how to manage time effectively and make sure all team members are on task.

VIII.3.5 Working in teams

Through this project, I learned more than I expected to about how to work effectively as a team. Not only does teamwork require great communication, but it requires compromise, substantial effort from each team member, and trust. When working as a team on projects, especially projects that last a long period of time, these factors are essential to keeping a team together and functioning. I believe our weekly client meetings were a driving factor in our team's success. Moreover, our ability to collaborate productively and trust each other with our respective work was essential to our accomplishments.

VIII.3.6 If you could do it all over again, what would you do differently?

If I could do this project over again, there are a couple things I would do differently. First of all, I would focus on coming up with creative solutions to the problems given to me by my client, as opposed to simply implementing their recommendations. As a programmer it is easy to just code what you're told to code; however, I believe part of being a programmer is coming up with creative solutions that complete your task as efficiently as possible. This is true especially when working with a client who does not have experience in programming. They may recommend some method for completing the tasks they hand you, when in reality you as a programmer may understand that a different method could be more efficient for solving your problem.

Another thing I would do differently is focus on having fun and working with my group to really make the project our own. Coming into capstone I felt relatively intimidated and stressed by the seriousness of the course. However, as I approach completion of the course, I see how the purpose of capstone is truly focused on my development and preparing me for work in the real world. Therefore, I would focus on trying to have fun with this project and explore some creative options that could improve our end product and provide me and my teammates with more knowledge in a different area of programming.

IX APPENDIX 1: ESSENTIAL CODE LISTING

The following code is the core of our webapp framework. Each webpage is constructed as a "view" in Django, shown in `views.py` below. Functionality for phase 2 is included as comments/pseudocode.

```
from django.http import HttpResponse
from django.shortcuts import render
from app.forms import *
from aerolyzer import *
from django.contrib.auth.decorators import login_required
from django.contrib.auth import logout
from django.views.decorators.csrf import csrf_protect
from django.http import HttpResponseRedirect
from django.template import RequestContext
from django.contrib.auth import authenticate
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login
from django.contrib import messages
```

```
from django.conf import settings
from django.core.files.storage import FileSystemStorage
from shutil import copyfile
import os
import pysolr
import time


def index(request):
    if request.method == 'POST':
        usr = request.POST['username']
        psswd = request.POST['password']
        user = authenticate(username=usr, password=psswd)
        if user is not None:
            login(request, user)
            return HttpResponseRedirect('gallery')
        else:
            return render(request, 'app/index.html', {'login_message' :
                'That␣username/password␣doesn\'t␣work!'},)
    return render(request, 'app/index.html', {'user': request.user,
        'is_index':True},)

def about(request):
    return render(request, 'app/about.html', {'user': request.user},)

def faq(request):
    return render(request, 'app/faq.html', {'user': request.user},)

def signup(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            user = User.objects.create_user(
            username=form.cleaned_data['username'],
            email=form.cleaned_data['email'],
            password=form.cleaned_data['password1']
            )
            return HttpResponseRedirect('signup_complete')
    else:
```

```
        form = RegistrationForm()
    return render(request,
    'app/signup.html',
    {'form': form, 'user': request.user, 'is_index':True,},
    )

def signup_complete(request):
    return render(request, 'app/signup_complete.html',{ 'user': request.user },)

@login_required
def gallery(request):
    return render(request,
    'app/gallery.html',
    { 'user': request.user },
    )

@login_required
def upload(request):
    if request.method == 'POST' and request.FILES['myfile']:
        myfile = request.FILES['myfile']
        fs = FileSystemStorage()
        filename = fs.save(myfile.name, myfile)
        uploadedFileUrl = fs.url(filename)
        request.session['filename'] = filename
        request.session['uploadedFileUrl'] = uploadedFileUrl
        verified = check_image("media/" + filename)
        if not verified['meetsRest']:
            os.remove("media/" + filename)
            return render(request,
            'app/upload.html',
            { 'user': request.user, 'filename': filename, 'uploadedFileUrl':
                uploadedFileUrl,
            'error_message': verified['error_message'],},)
        else:
            locExifData = verified["locExifData"]
            exifData = verified["exifData"]
            latitude = locExifData['gps_gpslatitude']
            d = float(latitude.values[0].num) / float(latitude.values[0].den)
            m = float(latitude.values[1].num) / float(latitude.values[1].den)
```

```
            s = float(latitude.values[2].num) / float(latitude.values[2].den)
            exifLat = d + (m / 60.0) + (s / 3600.0)
            if locExifData["gps␣gpslatituderef"].values[0] != "N":
                exifLat = 0 - exifLat
            longitude = locExifData["gps␣gpslongitude"]
            d = float(longitude.values[0].num) / float(longitude.values[0].den)
            m = float(longitude.values[1].num) / float(longitude.values[1].den)
            s = float(longitude.values[2].num) / float(longitude.values[2].den)
            exifLong = d + (m / 60.0) + (s / 3600.0)
            if locExifData["gps␣gpslongituderef"].values[0] != "E":
                exifLong = 0 - exifLong
            location = "%f,%f" % (exifLat, exifLong)
            exifData['location'] = location
            request.session['exifData'] = exifData
            return HttpResponseRedirect('retrieve')
    return render(request, 'app/upload.html', { 'user': request.user, },)

@login_required
def profile(request):
    return render(request,
    'app/profile.html',
    { 'user': request.user, },
    )

@login_required
def retrieve(request):
    uploadedFileUrl = request.session['uploadedFileUrl']
    filename = request.session['filename']
    exifData = request.session['exifData']
    location = exifData['location']
    if request.method == 'POST':
        weatherData = wunderData.get_data(location)
        if weatherData is None:
            os.remove("media/" + filename)
            return render(request,
             'app/retrieve.html',
            { 'user': request.user, 'exifData' : exifData, 'all_clear': False,
            'error_message': 'weather', 'filename': filename,
            'uploadedFileUrl': uploadedFileUrl,},)
```

```
        request.session['wunderData'] = weatherData
        # misrData = retrieve_misr_info(location)
        # if misrData is None:
        #   os.remove("media/" + filename)
        #   return render(request,
        #     'app/retrieve.html',
        #   { 'user': request.user, 'error_message': 'satellite' },
        #     )
        # request.session['misrData'] = misrData
        # return render(request,
        # 'app/retrieve.html',
        # { 'user': request.user, 'exifData' : exifData,
        # 'wunderData': wunderData, 'misrData': misrData, 'all_clear': True, },
        # )
        return render(request,
        'app/retrieve.html',
        { 'user': request.user, 'exifData' : exifData,
        'wunderData': weatherData, 'misrData': 'misr here', 'all_clear': True,
        'filename': filename, 'uploadedFileUrl': uploadedFileUrl,},
        )

    return render(request,
    'app/retrieve.html',
    { 'user': request.user, 'filename': filename, 'uploadedFileUrl': uploadedFileUrl,
    'all_clear': False,},
    )

@login_required
def results(request):
    uploadedFileUrl = request.session['uploadedFileUrl']
    filename = request.session['filename']
    exifData = request.session['exifData']
    weatherData = request.session['wunderData']
    # misrData = request.session['misrData']
    # aerosol = coreAlgorithmHere(exifData, wunderData, misrData)
    username = request.user.username
    unique = str(int(time.time()))
    solrFilename = username + "-" + unique + "_" + filename
    solr = pysolr.Solr('http://localhost:8983/solr/aerolyzer', timeout=10)
```

```
    solr.add([
        {
            "filename": solrFilename,
            "exif": exifData,
            #"misr": misrData,
            "wunder": weatherData,
            #"results": aerosol,
            "username": username,
        },
    ])
    newFilename = os.path.abspath('../..') + "/installDir/" + unique + "_" + filename
    copyfile(os.getcwd() + "/media/" + filename, newFilename)
     # return render(request,
    # 'app/results.html',
    # { 'user': request.user, 'aerosol': aerosol,
    # 'image': retrievedSolrImg},
    # )
    return render(request,
    'app/results.html',
    { 'user': request.user, 'filename': filename, 'uploadedFileUrl': uploadedFileUrl,},)

def logout_page(request):
    logout(request)
    return HttpResponseRedirect('/app')
```