Plant Attribute Extraction

Ronald Batista

Directory Manipulation

4 December 2022

Table of Contents

List of Figures

1. Subsystem Overview

1.1. Subsystem Description

The subsystem contains code created to analyze the SSH directories for any existing generated data to allow the user to download the data directly from the directory if it is already saved within the database. However, the data would be deleted after a specific duration of time. This to prevent the database from piling up unused storage over time. A generated directory should only stay within the database for only about 2 weeks before being deleted. The subsystem also ensures that the European Petroleum Survey Group (EPSG) matched with the Orthomosaic to properly project the generated files on top of the Orthomosaic on the QGIS software.

1.2. Current State of the Subsystem

Currently, the code named checkZipStatus.py, checkZipStatus.php, and the ajax call in main.js is 100% functional and has no noticeable bugs as of the day of the demo. The EPSG code found in generateZip.py and generate_dat_files_new.py is 100% functional within the system with no noticeable bugs as of the day of the demo. The code named deleteTempResults.py, deleteTempResults.php, and the ajax call in main.js is functional for when the timer is set to at most 1 hour. When validating the deletion for two weeks, there were some cases where the code was functional and others would not completely delete as such.

1.3. Future Improvements

Currently, the deletion of the files can be optimized to maybe inform the user better on what files are existing within the database and allow them to download directly without having to generate to discover what directories are existing. A list could be added when the user selected the sensor in the project to then pop up a list of the files existing. These improvements were not added to the project since the main scope of the project was to optimize the website for the merging of multiple files and this subsystem was to help optimize the addition of that feature.

2. Void model

2.1. Model Requirements

The code is made up of PHP, Python, and JS code that flows with the other subsystems as shown in figure 1. The section highlighted in green is the directory management subsystem. The code analyzes what the user inputted and responds based on the user input. Based on whether the directory has already existed or not, it

then passes to attribute modification and storage to then generate and download the selected files and attributes. In the code, the .ajax from the JS function will call the PHP that will then pass to the Python file in order for the code to function.

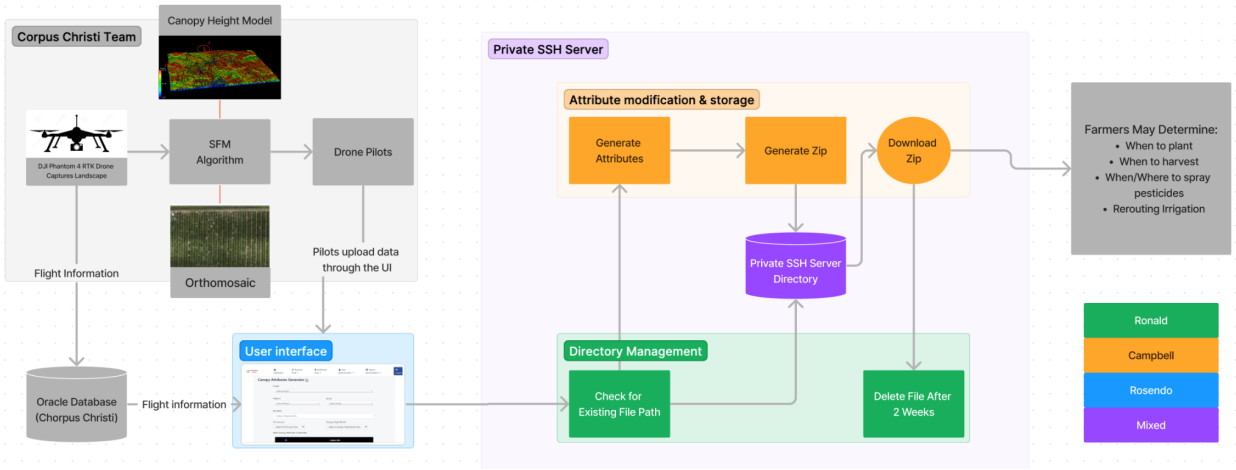

Figure 1: Process of CheckZipStatus and DeleteTempResults

The EPSG code is made of variables in the Python code that allow the EPSG value to be taken in when the EPSG is added at the creation of the project on the website.

2.2. Code Structure

2.2.1. Python

The code structure for checkZipStatus.py analyzes the directory based on the user input and grabs the complete file path. An if statement then checks if the directory being generated by the user input is similar to an existing directory in the database, it will pass to say that the data is already generated and (as will be explained in the PHP section as well) will be grabbed by the PHP code to utilize in the ajax call in JS. The code for deleteTempResults has a smaller format for grabbing the file path with an added timer to last until the code will then need to be deleted. An if statement will then need to be called to then delete the file if it is existing.

```python
import sys
import os
import shutil
import time

def main():
    zip_file_path = sys.argv[1] #grabs system arguments containing the path to the temp results dirtectory and zipfile downloaded by the client
    #print("Selected File Path noExt: ", zip_file_path) #code here is similar to setup of deleteTempResults.py
    cc_ckd = sys.argv[2]
    exg_ckd = sys.argv[3]
    ch_ckd = sys.argv[4]
    cv_ckd = sys.argv[5]
    attributes = []
    if (cc_ckd == 'true'):
        attributes.append("cc_boundary")
    if (exg_ckd == 'true'):
        attributes.append("exg_boundary")
    if (ch_ckd == 'true'):
        attributes.append("ch_boundary")
    if (cv_ckd == 'true'):
        attributes.append("cv_boundary")
    ortho_temp = sys.argv[6].split(',')
    ortho_temp = sorted(ortho_temp)
    project = sys.argv[7].split("::")
    project_name = project[0]
    index = 0
    print("zip: ", zip_file_path)
    print("cc: ", cc_ckd)
    print("exg: ", exg_ckd)
    print("ch: ", ch_ckd)
    print("cv: ", cv_ckd)
    print("ortho: ", ortho_temp)
    print("project: ", project)
    chm_counter = []
    print("attributes: ", attributes)
```

```python
    for i in ortho_temp:
        for j in attributes:
            file_path = "/var/www/html/uas_data/download/product/" + project_name + '/'
            print("file name: ", file_path)
            if (os.path.exists(file_path)):
                print(j,' exists')
                if (j == 'ch_boundary' or j == 'cv_boundary'):
                    print("A1")
                    temp_file_path_csv = file_path + '/' + j + '_' + i + '.csv'
                    temp_file_path_xlsx = file_path + '/' + j + '_' + i + '.xlsx'
                    temp_file_path_geoJSON = file_path + '/' + j + '_' + i + '.geoJSON'
                    temp_file_path_shp = file_path + '/' + j + '_' + i + '.shp'
                    print("A2")
                    if (os.path.exists(temp_file_path_csv)):
                        print("A3")
                        chm_counter.append(i.split("_")[0])
                        print("A3b")
                    elif (os.path.exists(temp_file_path_xlsx)):
                        print("A4")
                        chm_counter.append(i.split("_")[0])
                    elif (os.path.exists(temp_file_path_geoJSON)):
                        print("A5")
                        chm_counter.append(i.split("_")[0])
                    elif (os.path.exists(temp_file_path_shp)):
                        print("A6")
                        chm_counter.append(i.split("_")[0])
                    print("B")
                else:
                    print("else")
                    return;
                if (index == 0):
                    temp_file_path = file_path + '/' + j + '_' + i + '.csv'
                    if (os.path.exists(temp_file_path)):
                        print("csv")
                    temp_file_path = file_path + '/' + j + '_' + i + '.xlsx'
                    if (os.path.exists(temp_file_path)):
                        print("xls")
                    temp_file_path = file_path + '/' + j + '_' + i + '.geojson'
                    if (os.path.exists(temp_file_path)):
                        print("geoJSON")
                    temp_file_path = file_path + '/' + j + '_' + i + '.shp'
                    if (os.path.exists(temp_file_path)):
                        print("shp")
                index += 1
```

```
    print("C")
    print(chm_counter)
    if ((len(chm_counter) == len(ortho_temp)) or (len(chm_counter) == 0)):
        print("all")
    bool_check = 0 #case is to interpret the result so that it can be grabbed for other code
    #attribute_path = zip_file_path +
    if (os.path.exists(zip_file_path)): #difference lies here where the file is not deleted
        bool_check = 1
        #print("The temp results directory and zip file have already been generated, you can download the generated results.", bool_check)
        print("The temp results directory and zip file have already been generated, you can download the generated results.")
    else:
        bool_check = 0
        print("The temp results directory and zip file do not exist!", bool_check)

if __name__ == "__main__":
    main()
```

Figure 2: checkZipStatus.py code

```
    #t = 604800  # a week in seconds
    t = 120 #timer test
    while t:  #loop creates timer to keep data in data in database for a week before deleting
        mins, secs = divmod(t, 60)
        timer = '{:02d}:{:02d}'.format(mins, secs)
        time.sleep(1)
        t -= 1 # reduces t variable
```

Figure 3: Timer in deleteTempResults.py

The EPSG code consists of variables added that read in the value of the EPSG taken from the orthomosaic added in the website when creating the project. The variable is located within generateZip.py and generate_dat_files.py. Originally, the code was in the PHP files for these Python files, but it was revealed that it was not necessary for the variables to be there.

```
selected_orthomosaic_EPSG = selected_orthomosaic[2]
```

Figure 4: EPSG Variable

2.2.2. PHP

The PHP code between checkZipStatus and deleteTempResults has the same code structure since in both occasions, the code's main function is to call the python code and relay the data to the website.

```php
<?php
#PHP code that grabs the path to the zip file and the temporary results directory.
#The code then calls to a python file (CheckZipStatus.py) which then checks the status of the zip.
    $zip_file_path = $_GET["zipfile_path"];
    $zip_file_path_noExt = $_GET["zipfile_path_noExt"];

    $returningArray = array();
    array_push($returningArray, $zip_file_path);
    array_push($returningArray, $zip_file_path_noExt);

    # Call Python code with field information to check the Zip file and temp results directory.
    $checkZip_command = "python3 /var/www/html/uas_tools/canopy_attribute_generator/Resources/Python/CheckZipStatus.py $zip_file_path_noExt $zip_file_path";
    array_push($returningArray, $checkZip_command);
    $result = shell_exec($checkZip_command);
    if($result){                                    // If there is a result push result to array being passed back in JSON format
      # echo "Python has been executed!";
      array_push($returningArray, $result);
       }else{
         array_push($returningArray, "ERROR: Was unable to call to the CheckZipStatus python executable!"); // Else, append Error on to returning JSON object.
       }

    echo json_encode($returningArray);     //return returning array regarding this call in JSON format.
    die();

?>
```

Figure 5: checkZipStatus.php code structure

2.2.3. JS

The JS was similar as well since the function, much like the PHP code, was for execution purposes. This was implemented by using the .ajax command to utilize the PHP code. The execution of the code is shown in the console when the phrase “The checkZipStatus file was called!” or “The deleteTempResults file was called!” This will be shown in the validation section of the report. The checkZipStatus included alerts to let the user know of the existing files and which types of files they could download and generate.

```
$.ajax({                                          //AJAX call to generateResults.php
      async: false,
      url: "Resources/PHP/checkZipStatus.php",
      datatype: "POST",
      data: {                                             // Data being sent to PHP file
        zipfile_path_noExt: zip_file_path_noExt,
        cc_ckd: cc_checked,
        exg_ckd: exg_checked,
        ch_ckd: ch_checked,
        cv_ckd: cv_checked,
        ortho_temp: check_zip_ortho_temp,
        project: selected_project
      },
      beforeSend: function(){
        $("#loader").css({display: 'flex'});                    //Show loading screen
        $("#myProgress").show();
        $("#loader2").show();
      },

      success: function (response) {
        //console.log("response:")
        //console.log(response);
        chm_array = [];
        var temp = 0;
        for (let a = 0; a < attributes.length; a ++) {
          if (response.indexOf(attributes[a]) == -1) {
            return;
          }
        }
        var targetString = "The temp results directory and zip file have already been generated, you can download the generated results.";
        if(response.indexOf(targetString) > -1){
          //Error handling: If project/Ortho results do not exist, kill process
          //existingZip = true;
          files = [];
          if (response.indexOf('csv') > -1) {
            files.push('csv');
          }
          if (response.indexOf('xls') > -1) {
            files.push('xls');
          }
          if (response.indexOf('geoJSON') > -1) {
            files.push('geoJSON');
          }
          if (response.indexOf('shp') > -1) {
            files.push('shp');
          }
          console.log(files);
          if (response.indexOf('all') == -1) {
            orthomosaic_array = selected_ortho_temp.split(',');
            orthomosaic_array.sort();
            for (let o = 0; o < orthomosaic_array.length; o++) {
              if (response.indexOf(orthomosaic_array[o]) > -1) {
                chm_array.push(orthomosaic_array[o].split("_")[0]);
                console.log(chm_array);
                //alert(files);
              }
            }
            alert("The files listed already exist. You're free to download. " + files, " The Canopy Height and Volume only exist for these orthomosaics: " + chm_array);
          } else {
            alert("The file types listed already exist. You're free to download them. " + files);
          }

          //alert("The files listed already exist. You're free to download", files);
          $('#loader').hide();
          $('#loader2').hide();                           //When complete, hide the Loading screen.
          $('#myProgress').hide();
          temp = 1;
          store(temp);
          //generateZip();
          return;
        }
        //console.log("The checkZipStatus file was called!");
      },
```

Figure 6: checkZipStatus JS code in main.js

3. Validation

To validate the subsystem, the code needed to show functionality as the website was being used. It also had to show the code running in the console of the website. For all areas of the subsystem, the code created for the project was in working order in conjunction with the code created by other members of the team.

3.1. Validating EPSG

To confirm that the EPSG code was running correctly within the console, what is needed to be seen in the console is the orthomosaic file path as well as the EPSG being created as a part of the console code. The EPSG is to be referenced in the QGIS software to properly match within the projection. The figure below shows the EPSG result for the Amarillo project.

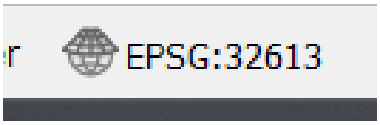

Figure 7: EPSG Validation

In the second console result, the EPSG value for the project is shown at the end of the print statement.

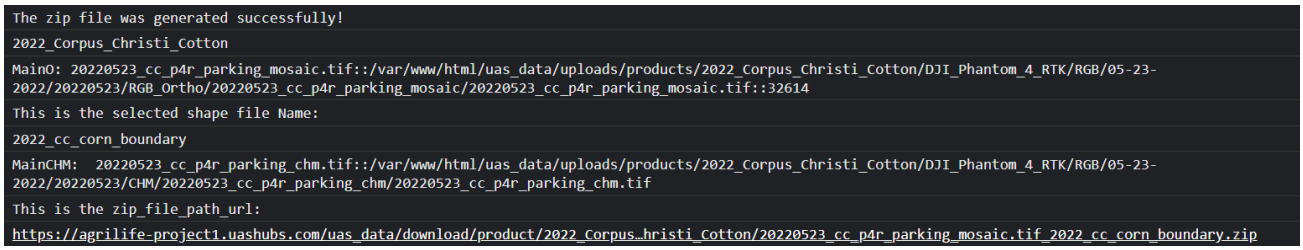

Figure 8: generateZip EPSG Validation

3.1.3. Validation Results

As shown in figures 7 and 8, the EPSG result is generated as the values 32613 and 32614 for Amarillo and Corpus Christi respectively. It is shown in the results the file path of the Orthomosaic for each figure.

3.2. Validating checkZipStatus

To validate checkZipStatus, the console has to output the phrase “The checkZipStatus file was called!” as well as show the php file running with the output of the Python code in the Response tab of the inspect. Without looking at the console, the user will receive a pop-up that lets the user know the file exists as well as the files they may be able to download.

Figure 9: checkZipStatus Validation

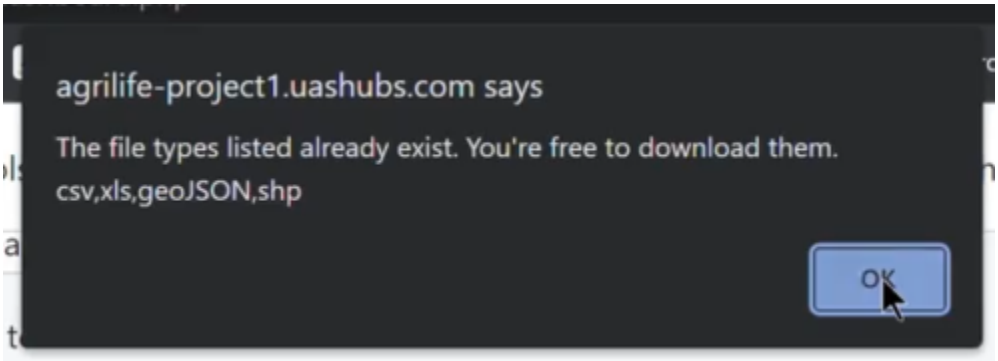

Figure 10: User Notification of Existing Files

3.2.3. Validation Results

As shown in the figures above, the php code is highlighted on the left of the figure with the phrase as described shown at the bottom of the figure. The Response tab is open to show the execution of the Python code. checkZipStatus is running within the background of the website. The notification also shows the user the status of the directory and is able to download the files notified successfully.

3.3. Validating deleteTempResults

To validate deleteTempResults, the console has to output the phrase “The deleteTempResults file was called!” as well as show the php file being run with the output of the Python code in the Response tab of the inspect. The timing of the code was shown during the demo. While the sponsor requested for two weeks, we tested for two minutes during the demo (the code is able to run for two weeks). To also validate, I opened the remote database to scroll through the file path to see if the file was deleted when the prompt came up.

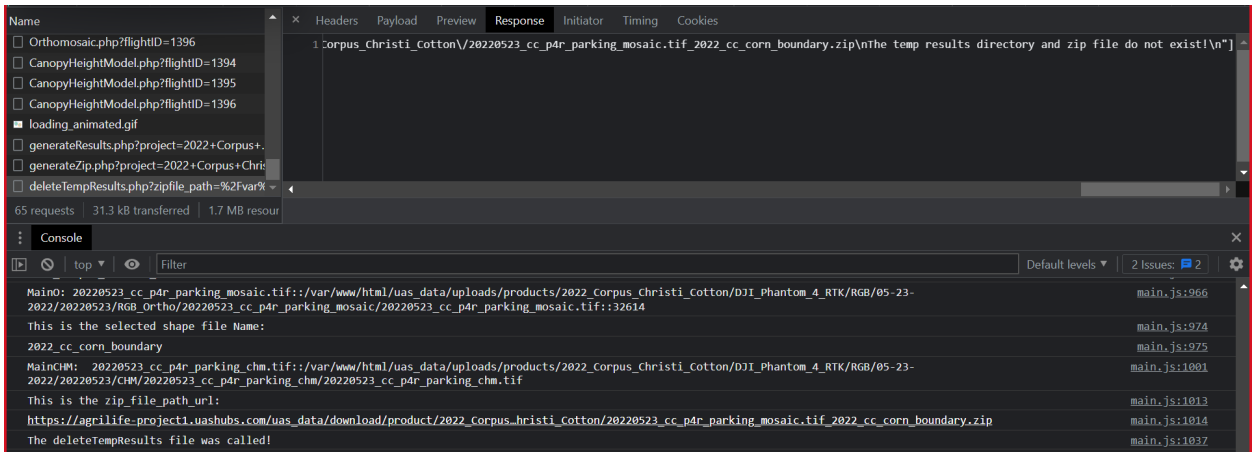

Figure 11: deleteTempResults Validation

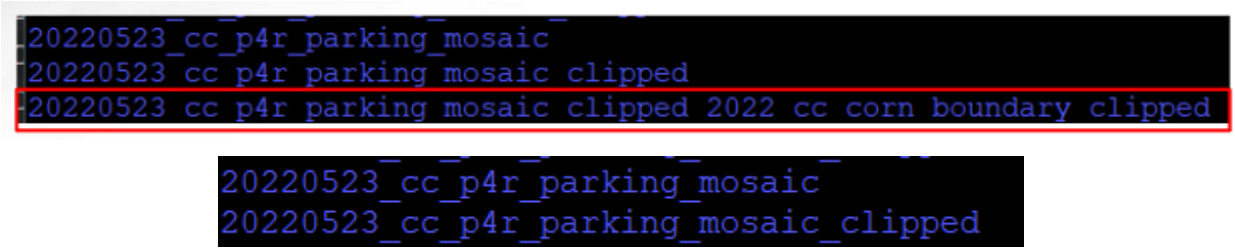

Figure 12: Directory before and after the Deletion of the Directory

3.3.3. Validation Results

As shown in the figure above, the php code is highlighted on the left of the figure with the phrase as described shown at the bottom of the figure. The Response tab is open to show the execution of the Python code. deleteTempResults is running within the background of the website. The database also showed that the data deleted successfully when prompted.

Appendix A: Acronyms and Abbreviations

EPSG	European Petroleum Survey Group
JS	JavaScript
PHP	Hypertext Prepocessor
HTML	HyperText Markup Language

Appendix B: Definition of Terms

- Orthomosaic: An image with a high resolution that combines miniature photos to create the eventual image
- Variable: A value identified by a title made by the user. The value could be text, numerical, or some function of other variables.

Appendix C: Important Links

- GitHub Link: https://github.com/RonBatista/ECEN404_Capstone