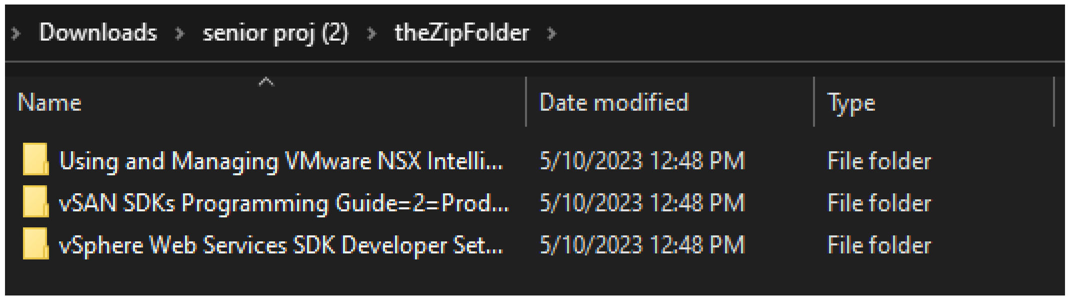
Project Lighthouse Documentation

Administrator Script User Manual:

Requirements:

- Python 3.0+
- c/lighthouse/documents directory exists
- A zip file containing product documentation
    - Each product must contain a metadata.xml file
    - We can handle zip files that look like this:

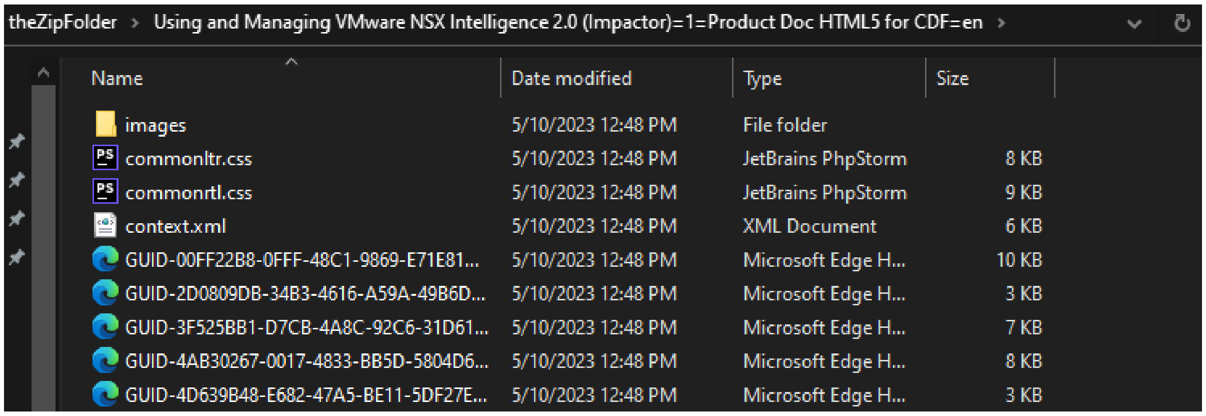


Downloads > senior proj (2) > theZipFolder >

Name	Date modified	Type
Using and Managing VMware NSX Intelli...	5/10/2023 12:48 PM	File folder
vSAN SDKs Programming Guide=2=Prod...	5/10/2023 12:48 PM	File folder
vSphere Web Services SDK Developer Set...	5/10/2023 12:48 PM	File folder

    - Where each folder inside the zip folder contains documentation for a particular product

- We can also handle just 1 product at a time with no folder which would look like this (this folder also needs to include a index.html file):



theZipFolder > Using and Managing VMware NSX Intelligence 2.0 (Impactor)=1=Product Doc HTML5 for CDF=en >

Name	Date modified	Type	Size
images	5/10/2023 12:48 PM	File folder	
commonltr.css	5/10/2023 12:48 PM	JetBrains PhpStorm	8 KB
commonrtl.css	5/10/2023 12:48 PM	JetBrains PhpStorm	9 KB
context.xml	5/10/2023 12:48 PM	XML Document	6 KB
GUID-00FF22B8-0FFF-48C1-9869-E71E81...	5/10/2023 12:48 PM	Microsoft Edge H...	10 KB
GUID-2D0809DB-34B3-4616-A59A-49B6D...	5/10/2023 12:48 PM	Microsoft Edge H...	3 KB
GUID-3F525BB1-D7CB-4A8C-92C6-31D61...	5/10/2023 12:48 PM	Microsoft Edge H...	7 KB
GUID-4AB30267-0017-4833-BB5D-5804D6...	5/10/2023 12:48 PM	Microsoft Edge H...	8 KB
GUID-4D639B48-E682-47A5-BE11-5DF27E...	5/10/2023 12:48 PM	Microsoft Edge H...	3 KB

- No internet connection needed

Initialization:

- Drag the lighthouse folder into your C drive
- Double click the lighthouse.py script

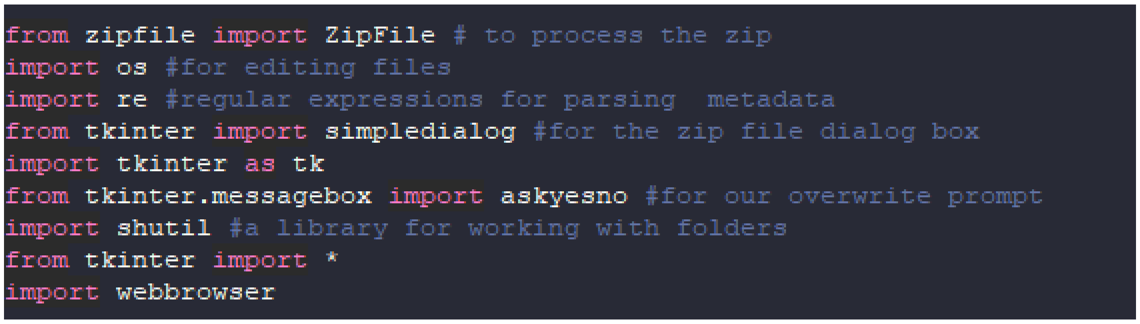
Usage:

- **Uploading**:
    - Run the script
    - Click the upload button
    - Copy and paste the path to your zip file into the prompt
    - Hit OK
    - Directories will be built, if they do not already exist
    - Supports multiple versions/languages of the same product
    - Zip file can be filled with product folders or can be 1 singular product so long as the metadata.xml file exists
- **Overwriting**:
    - If the product documentation already exists while uploading, you'll receive a file already exists prompt
    - Click on the button provided
    - It will ask you if you wish to override
    - If you override, it will delete the content that already exists and replace it with the override content
    - If you choose not to override, it will keep the old content
    - Will receive a prompt for each file path that is already existing
- **Deleting**:
    - Run script

- Select the delete button
- Will be provided all of the current content paths in the documents folder
- Can select single or multiple entries to be deleted
- Once selected, hit the delete button
- Content will be deleted from the documents folder

Administrator Script Developer Manual:

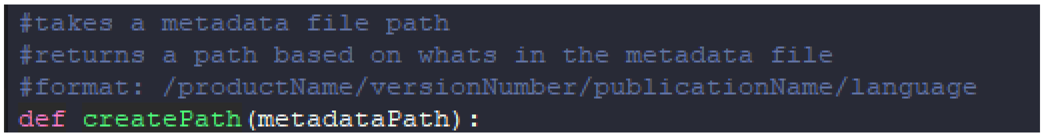
- **Imports:**
    - All imports are from the python standard library



```python
from zipfile import ZipFile # to process the zip
import os #for editing files
import re #regular expressions for parsing  metadata
from tkinter import simpledialog #for the zip file dialog box
import tkinter as tk
from tkinter.messagebox import askyesno #for our overwrite prompt
import shutil #a library for working with folders
from tkinter import *
import webbrowser
```

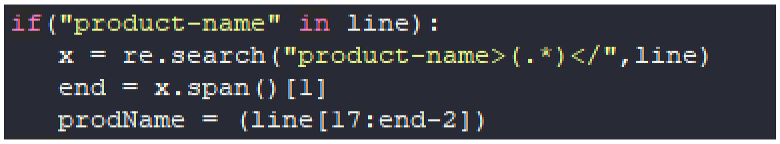
- Our zipfile is used in order to do the extraction of the zip folder
- The os library is used heavily mainly for directory and file manipulation
- The re library is for regular expressions in python, used for searching the metadata file
- Tkinter imports are for the GUI elements including buttons, menus, and panes
- shutil is similar to os but is for more high level operations like recursively deleting directory trees, etc.
- Webbrowser allows us to open up the initial landing page

- **createPath(metadataPath):**



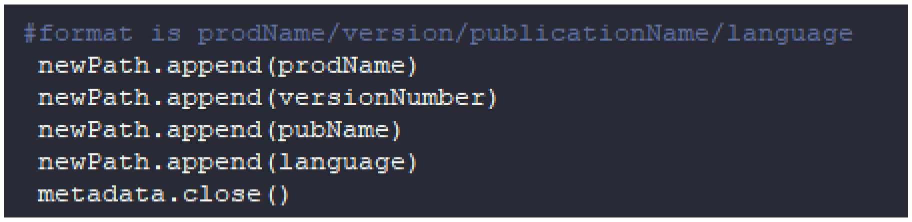
```
#takes a metadata file path
#returns a path based on whats in the metadata file
#format: /productName/versionNumber/publicationName/language
def createPath(metadataPath):
```

- Parameters:
    - metadataPath = the path to a metadata.xml file
- Returns:
    - A directory path that we will later use for building the directory ex. productName/versionNumber/publicationName/language
- The function opens up the metadataPath parameter in read mode
- It reads all of the lines in the file using .readlines() and then loops through each line
- Also creates string variables for each part of the returning path
    - **IMPROVEMENT:** turn this into a dictionary instead of string variables
- Loop through each line using regex to find each of the tags we are looking for, strip the string to get rid of the xml tags
    - **IMPROVEMENT:** use a xml file reader library instead of regular expressions (https://docs.python.org/3/library/xml.dom.minidom.html)



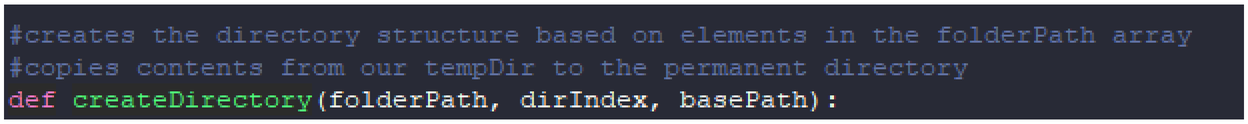
```
if("product-name" in line):
   x = re.search("product-name>(.*)</",line)
   end = x.span()[1]
   prodName = (line[17:end-2])
```

- The metadata.xml file doesn't always follow the same format, thus there is some exceptions to odd formatting
- Once looped through, return the path created



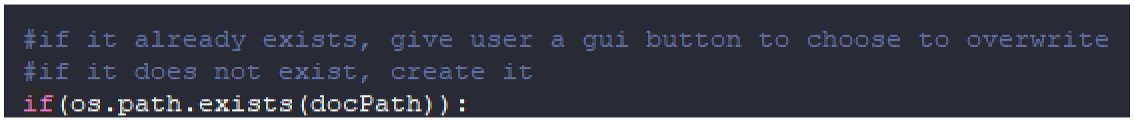
```
#format is prodName/version/publicationName/language
 newPath.append(prodName)
 newPath.append(versionNumber)
 newPath.append(pubName)
 newPath.append(language)
 metadata.close()
```

- ## createDirectory(folderPath, dirIndex, basePath):



```
#creates the directory structure based on elements in the folderPath array
#copies contents from our tempDir to the permanent directory
def createDirectory(folderPath, dirIndex, basePath):
```

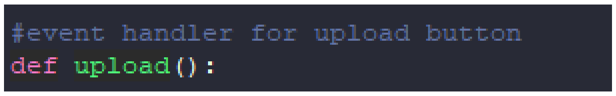
- Parameters:
    - folderPath = a list containing the path we generated earlier ex.['VMware vSAN', '1.0', 'vSAN SDKs Programming Guide', 'en']
    - dirIndex = the index of the temporary directory where 0…n represents the number of directories inside the temporary directory
    - basePath = the zip folder of the files aka the source
- We start by opening up our index files that are used for the front end website, index.html is used on the side panel, index2.html is used on the document view
- We build a docPath which will be used later to see if we need to overwrite
- The overwrite is checked for here:



```
#if it already exists, give user a gui button to choose to overwrite
#if it does not exist, create it
if(os.path.exists(docPath)):
```

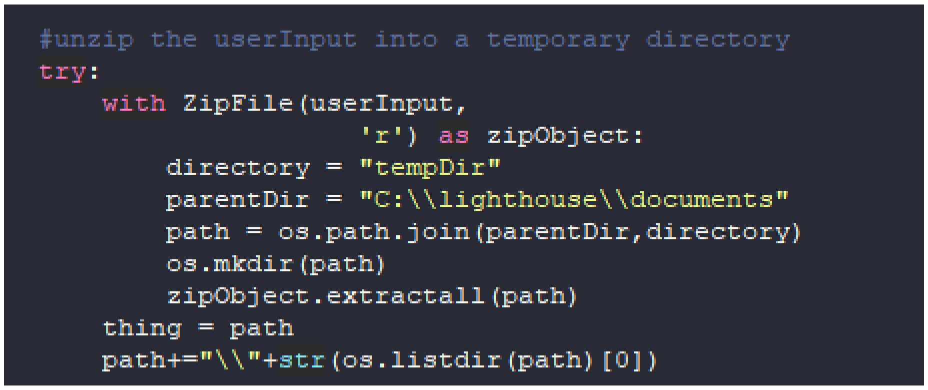
- This will prompt the user to submit an answer via Windows pop-up
- If the user answers yes:
    - Get to the end of the docPath
    - Delete all of the content out of it
    - Copy over the incoming content
- If the user answers no, do nothing
- If the content doesn't already exist:
    - Iterate through folderPath
    - Construct a new directory
    - Copy over the content from tempDir
    - Write html code to index.html using the docPath and display name
    - Also write to index2 which takes each products index and accumulates it
        - **IMPROVEMENT:** it currently does not delete from index2.html when deleting, add in that functionality

- ## upload()



```
#event handler for upload button
def upload():
```

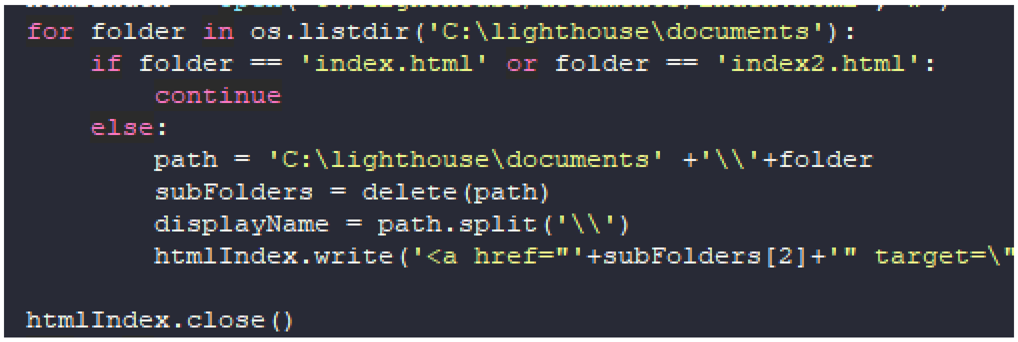
  - This is an event handler for the upload button
  - We use the tkinter simpleDialog to get the zip file path
  - We also use ZipFile here in order to unzip the file and copy it into tempDir



```
#unzip the userInput into a temporary directory
try:
    with ZipFile(userInput,
                    'r') as zipObject:
        directory = "tempDir"
        parentDir = "C:\\lighthouse\\documents"
        path = os.path.join(parentDir,directory)
        os.mkdir(path)
        zipObject.extractall(path)
    thing = path
    path+="\\"+str(os.listdir(path)[0])
```

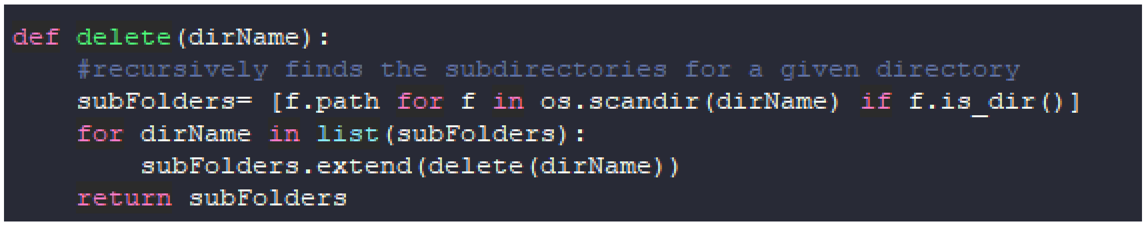
  - Catch an exception if the user inputs something bad
    - **IMPROVEMENT:** Currently they'll have to restart the program after submitting an incorrect file location, make it to where that doesn't happen
  - Next we have 2 cases
    - If its a folder inside of a folder we loop through the multiple different folders calling both our createPath() and createDirectory()
    - If it's just a singular product with a metadata file we instead call the createDirectory function with a negative directory index
    - Remove the temporary directory after processing

- ## delete(dirName) & dirHelper():
    - deleteHelper is the event handler for the delete button
    - It creates the tkinter listbox that we will use to display what the user can display
    - selected_item() is the event handler for the delete selected button
    - When the button is clicked it finds all of the entries the user has selected via listbox.curselection() and loops through them
    - It then adjusts the index.html file, only writing to it the things that the user didn't select



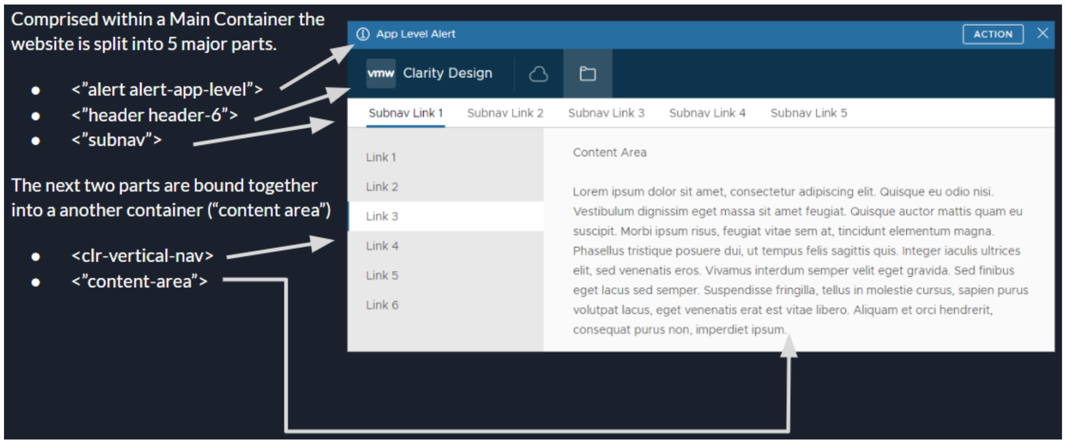
```
for folder in os.listdir('C:\lighthouse\documents'):
    if folder == 'index.html' or folder == 'index2.html':
        continue
    else:
        path = 'C:\lighthouse\documents' +'\\'+folder
        subFolders = delete(path)
        displayName = path.split('\\')
        htmlIndex.write('<a href="'+subFolders[2]+'" target=\"

htmlIndex.close()
```

- Then we use delete() which recursively finds all of the subdirectories and returns them to be deleted



```
def delete(dirName):
    #recursively finds the subdirectories for a given directory
    subFolders= [f.path for f in os.scandir(dirName) if f.is_dir()]
    for dirName in list(subFolders):
        subFolders.extend(delete(dirName))
    return subFolders
```

Website Client Information:

- The framework to create the website was mainly borrowed from the clarity design documentation. It explains how to best utilize the html/css framework. (https://clarity.design/documentation/get-started)
- The best way to view and understand the html code is to picture it into 7 parts. The first two being style and scripts. The next three are alert level, subnav, and header. The last two as vertical nav and content area.
- 

Comprised within a Main Container the website is split into 5 major parts.

- <"alert alert-app-level">
- <"header header-6">
- <"subnav">

The next two parts are bound together into a another container ("content area")

- <clr-vertical-nav>
- <"content-area">

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque eu odio nisi. Vestibulum dignissim eget massa sit amet feugiat. Quisque auctor mattis quam eu suscipit. Morbi ipsum risus, feugiat vitae sem at, tincidunt elementum magna. Phasellus tristique posuere dui, ut tempus felis sagittis quis. Integer iaculis ultrices elit, sed venenatis eros. Vivamus interdum semper velit eget gravida. Sed finibus eget lacus sed semper. Suspendisse fringilla, tellus in molestie cursus, sapien purus volutpat lacus, eget venenatis erat est vitae libero. Aliquam et orci hendrerit, consequat purus non, imperdiet ipsum.

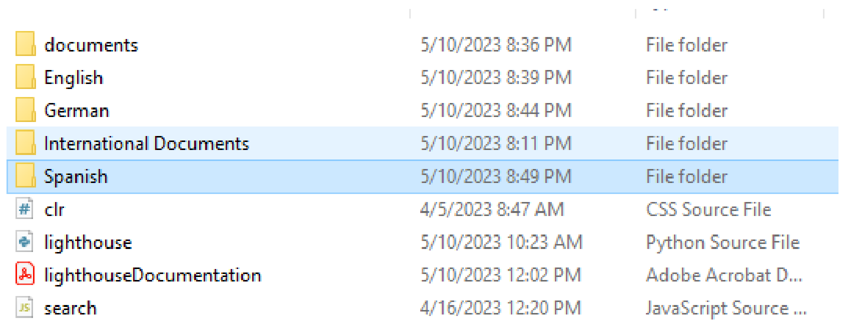
- This code would look like this

```
<div class="main-container">
    <div class="alert alert-app-level">
        ...
    </div>
    <header class="header header-6">
        ...
    </header>
    <nav class="subnav">
        ...
    </nav>
    <div class="content-container">
        <div class="content-area">
            ...
        </div>
        <clr-vertical-nav>
            ...
        </clr-vertical-nav>
    </div>
</div>
```

- The next major part of the code is Hypertext Inline Presentation. So to present our documentation we utilize Inline Frame Tags. They allow us to embed another document (pdf, xml, html) within another html file. Specifically in this instance we place it within the "Content Area" tags on the right side of the content area class.

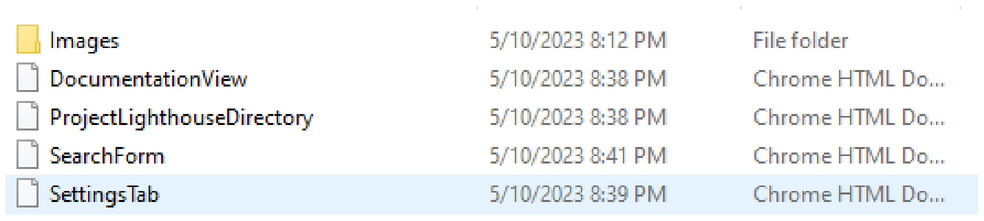
- These major concepts are what allow us to serve up the documentation.

Front End Structure:



documents	5/10/2023 8:36 PM	File folder
English	5/10/2023 8:39 PM	File folder
German	5/10/2023 8:44 PM	File folder
International Documents	5/10/2023 8:11 PM	File folder
Spanish	5/10/2023 8:49 PM	File folder
clr	4/5/2023 8:47 AM	CSS Source File
lighthouse	5/10/2023 10:23 AM	Python Source File
lighthouseDocumentation	5/10/2023 12:02 PM	Adobe Acrobat D...
search	4/16/2023 12:20 PM	JavaScript Source ...

-
- The front end structure is broken down into 5 major folders and 4 adjoining files.
- Documents and International Documents store the documentation as well as the indexes that are created.
- Clr Css source file is the Clarity Design Systems package so we can utilize it offline.
- The Lighthouse.py code is what generates the indexes and is used for the backend.
- The Lighthouse documentation is what you are currently using.
- The search javascript file was previously used to search through files on the webpage. It is deprecated for the lookup table for the english folder, but is still in use for the german and spanish folder.
- The English, German and Spanish folder are what house the html documents for the website as well as the images used for the site. (an example is shown below)



Images	5/10/2023 8:12 PM	File folder
DocumentationView	5/10/2023 8:38 PM	Chrome HTML Do...
ProjectLighthouseDirectory	5/10/2023 8:38 PM	Chrome HTML Do...
SearchForm	5/10/2023 8:41 PM	Chrome HTML Do...
SettingsTab	5/10/2023 8:39 PM	Chrome HTML Do...

Future Developments (Front End):

- Currently the German and Spanish Implementations utilize hardcoded pdf files because they don't fit our file type requirements. This can be solved by either receiving file types that fit our requirements or by creating a way for these different styled files to work. Without that specific file type we also could not generate the indexes for either the document view or the file lookup.
- The addition of more files to fill out our content management system would also better showcase our work.
- Creating a better search engine instead of the simplified file lookup would be a good way to further develop the system.
- The Document View tab needs some more work, we could not decide how to implement it due to it being very similar in nature to the dashboard. So we decided to just show off what a split view of the documentation could look like. Another way this document view tab could be changed which we tinkered with would be to have the left side vertical nav target the iframe on the right. This would functionally do the same thing as the dashboard view, but it would look very smooth.