Implementation

Captain CyBeard: Neil Before Us

Ryan Breitenfeldt | Noah Farris
Trevor Surface | Kyle Thomas

May 4, 2020

Washington State University Tri-Cities
CptS 423 Software Design Project 2

project_root/requirements.txt

```
asn1crypto==0.24.0
attrs==17.4.0
Automat==0.6.0
blinker==1.4
boto3==1.13.0
botocore==1.16.0
cachetools==4.0.0
certifi==2019.11.28
cffi==1.14.0
chardet==3.0.4
click==7.1.1
colorama==0.3.7
configobj==5.0.6
constantly==15.1.0
cryptography==2.4
Django==2.2.11
docutils==0.15.2
dropbox==9.4.0
Flask==1.1.2
google-api-python-client==1.7.11
google-auth==1.11.2
google-auth-httplib2==0.0.3
google-auth-oauthlib==0.4.1
httplib2==0.17.0
hyperlink==17.3.1
idna==2.9
incremental==16.10.1
itsdangerous==1.1.0
Jinja2==2.11.2
jmespath==0.9.5
jsonpatch==1.16
jsonpointer==1.10
jsonschema==2.6.0
keyring==10.6.0
keyrings.alt==3.0
MarkupSafe==1.1.1
netifaces==0.10.4
oauth2client==4.1.3
oauthlib==3.1.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.20
pycrypto==2.6.1
PyJWT==1.5.3
pyOpenSSL==17.5.0
pyserial==3.4
python-dateutil==2.8.1
python-debian==0.1.32
pytz==2019.3
```

```
pyxdg==0.26
requests==2.23.0
requests-oauthlib==1.3.0
requests-unixsocket==0.1.5
rsa==4.0
s3transfer==0.3.3
SecretStorage==2.3.1
service-identity==16.0.0
six==1.14.0
sqlparse==0.3.1
ssh-import-id==5.7
tqdm==4.45.0
Twisted==17.9.0
uritemplate==3.0.1
urllib3==1.25.8
Werkzeug==1.0.1
zope.interface==5.1.0
```

project_root/README.md

```
# Cloud Backup Application
Language:       Python 3.8 Django 2.2

Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt,
                        Noah Farris,
                        Trevor Surface,
                        Kyle Thomas

Class:              CptS 421/423 Fall 2019/Spring 2020

University:    Washington State University Tri-CIties

### Development

#### Making a Python Virtual Environment:

‘$ python -m venv <name of environment>‘ (I used cloud-backup-env)

Then add the folder it created to your .gitignore file

‘$ source <name of env folder>/bin/activate‘

You are now in the virtual environment anything you install will not be system wide

The first time you start:

‘$ pip install -r requirements.txt‘

To leave the environment:

‘$ deactivate‘

To add dependencies that you installed please do:

‘$ pip freeze > requirements.txt‘

Then commit the new requirements.txt to the repo.

#### Running the webserver:
Viewing on the same pc

‘$ python manage.py migrate‘

‘$ python manage.py runserver‘

Then visit: https://localhost:8000/cloud
```

Viewable from any pc on the network:

‘$ python manage.py runserver 0:8000‘

Adding a Cloud Platform
Add the file with the new cloud platform class into ‘cloud_download/platforms‘

Then import the file in ‘cloud_download/platforms/__init__.py‘

Finally, add the platform inside ‘cloud_download/views.py‘

project_root/cloud_backup/urls.py

```
"""cloud_backup URL Configuration

The ‘urlpatterns‘ list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('cloud_download.urls')),
    path('cloud/', include('cloud_download.urls')),
    path('admin/', admin.site.urls),
]
```

project_root/cloud_download/urls.py

```
#!/usr/bin/env python3

""" URL directs for cloud backup application
Application:    Cloud Backup
File:           /cloud_backup/cloud_download/urls.py
Description:    url paths
Language:       Python 3.8 Django 2.2
Dev Env:        Linux x64

Authors:        Ryan Breitenfeldt
                Noah Farris
                Trevor Surface
                Kyle Thomas
Class:          CptS 421/423 Fall '19 Spring '20
University:     Washington State University Tri-CIties
"""

from django.urls import path
from . import views

__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']


urlpatterns = [
    '''
    For Admins:
        The following paths need to be updated when using OAuth2.0 authentication. Depending upon th
        created two new 'paths' need to be created. A '[Platform_name]-auth-start', and a '[Platform
        same fashion below. otherwise the OAuth2.0 authentication will not work properly.
    '''
    path('', views.Index.as_view(), name='index'),
    path('dropbox-auth-start/', views.dropbox.dropbox_authentication_start),
    path('dropbox-auth-finish/', views.dropbox.dropbox_authentication_finish),
    path('google-auth-start/', views.google.GDriveDownloaded_authentication_start),
    path('google-auth-finish/', views.google.GDriveDownloaded_authentication_finish),
    path('files/', views.Files.as_view(), name='files'),
    path('aws_login/', views.Aws_Login.as_view(), name='aws_login'),
]
```

project_root/cloud_download/views.py

```
#!/usr/bin/env python3

""" URL views for cloud backup application
Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/views.py
Description:    Django Views / web pages
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-Cities
"""
import ast
from django.shortcuts import render, redirect
from django.views import View
from . import platforms
from .forms import AWS_AuthForm
from django import forms
import json
from .models import aws_data
import ast
from django.contrib import messages

__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']

'''
For Admins:
    When adding a new platform to the software if the platform SDK supports OAuth2.0 authentication
    Then the new platform class will need to be instantiated bellow along with the previous OAuth2.0
    class objects.
'''
global cloud
cloud = ""
dropbox = platforms.dropbox_script.DropBox()
google = platforms.gDriveDownloader.GDriveDownloader()


class Index(View):
    '''
    For Admins:
        When adding a new platform, the attribute platform will need to be updated with the new plat
        Additionally the post method of this class will need to be updated with elif similar to the
        of the post method. To finalize the post method, there also needs to be a redirect to '[plat
        To enable this as well the urls.py needs to be updated.
    '''
```

```
    index_template = 'cloud_download/index.html'
    platforms = ['google', 'dropbox', 'aws']

    def get(self, request):
        context = {'platforms': self.platforms}
        return render(request, self.index_template, context)

    def post(self, request):
        platform = request.POST['platform']
        global cloud
        if platform == 'google':
            print(platform)
            cloud = 'google'
            return redirect('google-auth-start/')
        elif platform == 'dropbox':
            cloud = 'dropbox'
            return redirect('dropbox-auth-start/')
        elif platform == 'aws':
            cloud = 'aws'
            return redirect('aws_login/')
        else:
            print("Unsupported platform")
            return redirect('index/')

        return redirect('aws_login/')

class Files(View):
    '''
    For Admins:
        The current functionality of the software requires a single level dict with the parameters {
        of dicts containing {'path':'', 'files':''} where 'files' is the name of the file without fo
        are implemented with a dict that allows mulilevel folder and file view, if the developer wou
        programming is required.

        The get method of this class will need to be updated with another elif to provide the necces
    '''

    template = 'cloud_download/files.html'
    success_template = 'cloud_download/success.html'
    context = {}
    def get(self, request):
        if cloud == 'dropbox':
            context = dropbox.dropbox_flat_dict
        elif cloud == 'google':
            context = google.GDriveDownloader_json
        elif cloud == 'aws':
            obj = aws_data.objects.first()
            key_id_object = aws_data._meta.get_field("aws_key_id")
            key_object = aws_data._meta.get_field("aws_key")
            aws_key_id = key_id_object.value_from_object(obj)
```

```
            aws_key= key_object.value_from_object(obj)
            aws = platforms.aws.aws(aws_key_id, aws_key)
            return render(request, self.template, {'files': aws.list_images_in_bucket()})

        return render(request, self.template, context)
    def post(self, request):
        '''
        For admins
            This method needs to be updated whe a new platform is added regardless of Oauth2 authent
            This is what calls the download procedures defined within the new platform class. Any ad
            information needed for downloading will need to be added to the context list and the inf
            transfered. Add an elif case for the new platform with a similar for loop consitant with
            if statements, then call the download funciton neccessary for the platform.
        '''

        context = {}
        user_selection = request.POST.getlist('box')
        files_to_download = []

        if cloud == 'aws':
            obj = aws_data.objects.first()
            key_id_object = aws_data._meta.get_field("aws_key_id")
            key_object = aws_data._meta.get_field("aws_key")
            aws_key_id = key_id_object.value_from_object(obj)
            aws_key= key_object.value_from_object(obj)
            aws = platforms.aws.aws(aws_key_id, aws_key)
            for file in user_selection:
                if file == 'on':
                    print('False')
                else:
                    file_name = ast.literal_eval(file)
                    aws.download_image('testbucket1293248523850923853', file_name['name'])
                    json_acceptable_string = file.replace("'", "\"")
                    files_to_download.append(json.loads(json_acceptable_string))
            context['files'] = files_to_download
            return render(request, self.success_template, context)

        elif cloud == 'dropbox':
            for file in user_selection:
                if file == 'on':
                    print('False')
                else:
                    file_name = ast.literal_eval(file)
                    files_to_download.append(file_name)
            context['files'] = files_to_download
            dropbox.dropbox_download_selected_entries(context['files'])
            return render(request, self.success_template, context)

        elif cloud == 'google':
            for file in user_selection:
                if file == 'on':
```

```
                    print('False')
                else:
                    files_to_download.append(ast.literal_eval(file))
            context['files'] = files_to_download
            if cloud == 'google':
                google.GDriveDownloader_files_to_download = files_to_download
                google.GDriveDownloader__download_File()
            return render(request, self.success_template, context)

'''
For Admins:
    The following class was implemented per the standard of AWS. If the new platform does not suppor
    then additional views will need to be created to allow the login parameters required by the plat
'''

class Aws_Login(View):
    template_name = 'cloud_download/aws_login.html'

    def get(self, request):
        form = AWS_AuthForm(request.POST)
        return render(request, self.template_name, {'form': form})

    def post(self, request):
        aws_data.objects.all().delete()
        form = AWS_AuthForm(request.POST)
        if form.is_valid():
            form.save()
            aws_key_id = form.cleaned_data.get('aws_key_id')
            aws_key = form.cleaned_data.get('aws_key')
            try:
                platforms.aws.aws(aws_key_id, aws_key).get_image_list()
                return redirect("/cloud/files/")
            except:
                messages.error(request, 'Amazon Web Services Key or Key ID is incorrect!')

        else:
            form = AWS_AuthForm()

        return render(request, self.template_name, {'form': form})

def index_redirect(request):
    return redirect('index/')
```

project_root/cloud_download/models.py

```
from django.db import models

class aws_data(models.Model):
    aws_key_id = models.CharField(max_length=500)
    aws_key = models.CharField(max_length=500)
```

project_root/cloud_download/forms.py

```
from django import forms
from .models import aws_data
from . import platforms

class AWS_AuthForm(forms.ModelForm):
    aws_key_id = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'AWS Key ID'}), label=
    aws_key = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'AWS Key'}), label='Key',
    class Meta:
        model = aws_data
        managed = False
        fields = ('aws_key_id', 'aws_key',)

    def clean_message(self):
            aws_key_id = self.cleaned_data.get('aws_key_id')
            aws_key = self.cleaned_data.get('aws_key')
            try:
                platforms.aws.aws(aws_key_id, aws_key).get_image_list()
            except:
```

project_root/cloud_download/apps.py

```
#!/usr/bin/env python3

""" App declaration for cloud backup application
Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/apps.py
Description:    Django App Config
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
"""

from django.apps import AppConfig

__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']


class CloudDownloadConfig(AppConfig):
    name = 'cloud_download'
```

project_root/cloud_download/static/cloud_download/aws.css

```
@import url(https://fonts.googleapis.com/css?family=Open+Sans);

body{
    font-family: 'Open Sans', sans-serif;
    background:#363636;
    margin: 0 auto 0 auto;
    width:100%;
    text-align:center;
    margin: 20px 0px 20px 0px;
  }

  p{
    font-size:12px;
    text-decoration: none;
    color:#ffffff;
  }

  h1{
    font-size:1.5em;
    color:#ddd;
    font: 'Open Sans', Arial, sans-serif;
  }

  h2{
    font-size:1.0em;
    color:#ddd;
    font: 'Open Sans', Arial, sans-serif;

  }
  .box{
    background:#484848;
    width:300px;
    border-radius:6px;
    margin: 0 auto 0 auto;
    padding:0px 0px 70px 0px;
    border: #2980b9 4px solid;
  }

  .aws_key_id{
    background:#ecf0f1;
    border: #ccc 1px solid;
    border-bottom: #ccc 2px solid;
    padding: 8px;
    width:250px;
    color:#AAAAAA;
    margin-top:10px;
    font-size:1em;
    border-radius:4px;
  }
```

```
.aws_key{
  border-radius:4px;
  background:#ecf0f1;
  border: #ccc 1px solid;
  padding: 8px;
  width:250px;
  font-size:1em;
}

.btn{
  background:#484848;
  width:125px;
  padding-top:5px;
  padding-bottom:5px;
  color:#ddd;
  border-radius:3px;
  border: #008CBA 2px solid;
  margin-top:20px;
  margin-bottom:20px;
  float:left;
  margin-left:80px;
  font-weight:800;
  font-size:0.8em;
}

.btn:hover{
  background: rgb(25, 63, 70);
}
```

project_root/cloud_download/static/cloud_download/aws_login_style.css

```
body{
    font-family: 'Open Sans', sans-serif;
    background:#363636;
    margin: 0 auto 0 auto;
    width:100%;
    text-align:center;
    margin: 20px 0px 20px 0px;
  }

  p{
    font-size:12px;
    text-decoration: none;
    color:#ffffff;
  }

  h1{
    font-size:1.5em;
    color:#ddd;
  }

  h2{
    font-size:1.0em;
    color:#ddd;
  }
  .box{
    background:white;
    width:300px;
    border-radius:6px;
    margin: 0 auto 0 auto;
    padding:0px 0px 70px 0px;
    border: #2980b9 4px solid;
  }

  .aws_key_id{
    background:#ecf0f1;
    border: #ccc 1px solid;
    border-bottom: #ccc 2px solid;
    padding: 8px;
    width:250px;
    color:#AAAAAA;
    margin-top:10px;
    font-size:1em;
    border-radius:4px;
  }

  .aws_key{
    border-radius:4px;
    background:#ecf0f1;
    border: #ccc 1px solid;
```

```
  padding: 8px;
  width:250px;
  font-size:1em;
}

.btn{
  background:#2ecc71;
  width:125px;
  padding-top:5px;
  padding-bottom:5px;
  color:white;
  border-radius:4px;
  border: #27ae60 1px solid;

  margin-top:20px;
  margin-bottom:20px;
  float:left;
  margin-left:80px;
  font-weight:800;
  font-size:0.8em;
}

.btn:hover{
  background:#2CC06B;
}

#btn2{
  float:left;
  background:#3498db;
  width:125px;  padding-top:5px;
  padding-bottom:5px;
  color:white;
  border-radius:4px;
  border: #2980b9 1px solid;

  margin-top:20px;
  margin-bottom:20px;
  margin-left:10px;
  font-weight:800;
  font-size:0.8em;
}

#btn2:hover{
background:#3594D2;
}
```

project_root/cloud_download/static/cloud_download/style.css

```
@import url(https://fonts.googleapis.com/css?family=Open+Sans);

/*Page styles*/
html { height: 100%; }

h1 {
    color: #ddd;
    text-align: center;
    font: 38px 'Open Sans', Arial, sans-serif;
}

h2 {
    color: #ddd;
    text-align: center;
    font: 15px 'Open Sans', Arial, sans-serif;
}

h3 {
  color: #ddd;
  text-align: center;
  font: 18px 'Open Sans', Arial, sans-serif;
}
body {
  height: 100%;
  margin: 0;
  background: #363636;
  align-items: center;
}
.outer {
    position: absolute;
    left: 30%;
}

.outer2 {
    position: absolute;
    left: 44%;
}

.outer3 {
  position: absolute;
  left: 27%;
}
.button {
    border: none;
    color: white;
    padding: 15px 45px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
```

```
  font: 16px 'Open Sans', Arial, sans-serif;
  margin: 8px 4px;
  cursor: pointer;
  vertical-align: top;
  border-radius: 3px
}

.button1 {
  background-color: #484848;
  color: #ddd;
  border: 2px solid #008CBA;
}
.button_index {
  border: none;
  color: white;
  padding: 15px 50px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font: 30px 'Open Sans', Arial, sans-serif;
  margin: 8px 4px;
  cursor: pointer;
  vertical-align: top;
  border-radius: 3px
}
.button2 {
  background-color: #484848;
  color: #ddd;
  border: 2px solid #008CBA;
}
.button1:hover{
  background: rgb(25, 63, 70);
  }
.button2:hover{
  background: rgb(25, 63, 70);
  }

  .btn{
    background:#484848;
    width:125px;
    padding-top:5px;
    padding-bottom:5px;
    color:#ddd;
    border-radius:3px;
    border: #008CBA 2px solid;
    margin-top:20px;
    margin-bottom:20px;
    float:left;
    margin-left:180px;
    font-weight:800;
    font-size:0.8em;
```

```
    }

    .btn:hover{
      background: rgb(25, 63, 70);
    }

.header {
    position: relative;
    top: 0;
}

.inner {
    position: relative;
}
.boxes {
  margin: auto;
  padding: 50px;
  background: #484848;
  border-radius: 7px;
}
.buttonHold {
    text-align: center;
}
/*Checkboxes styles*/
input[type="checkbox"] { display: none; }

input[type="checkbox"] + label {
  display: block;
  position: relative;
  padding-left: 35px;
  margin-bottom: 20px;
  font: 14px/20px 'Open Sans', Arial, sans-serif;
  color: #ddd;
  cursor: pointer;
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
}

input[type="checkbox"] + label:last-child { margin-bottom: 0; }

input[type="checkbox"] + label:before {
  content: '';
  display: block;
  width: 20px;
  height: 20px;
  border: 1px solid #6cc0e5;
  position: absolute;
  left: 0;
  top: 0;
  opacity: .6;
```

```
  -webkit-transition: all .12s, border-color .08s;
  transition: all .12s, border-color .08s;
}

input[type="checkbox"]:checked + label:before {
  width: 10px;
  top: -5px;
  left: 5px;
  border-radius: 0;
  opacity: 1;
  border-top-color: transparent;
  border-left-color: transparent;
  -webkit-transform: rotate(45deg);
  transform: rotate(45deg);
}

ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

li {
  font: 200 20px/1.5 Helvetica, Verdana, sans-serif;
  border-bottom: 1px solid #008CBA;
  text-align: center;
}

li:last-child {
  border: none;
}
.table {
display: table;   /* Allow the centering to work */
margin: 0 auto;
}
li a {
  text-decoration: none;
  color: #ddd;
  display: block;
  width: 600px;
  text-align: center;
  -webkit-transition: font-size 0.3s ease, background-color 0.3s ease;
  -moz-transition: font-size 0.3s ease, background-color 0.3s ease;
  -o-transition: font-size 0.3s ease, background-color 0.3s ease;
  -ms-transition: font-size 0.3s ease, background-color 0.3s ease;
  transition: font-size 0.3s ease, background-color 0.3s ease;
}

li a:hover {
  font-size: 30px;
}
```

project_root/cloud_download/templates/cloud_download/aws_login.html

```
<!--
file page template for cloud backup application

Application:     Cloud Backup
File:                /cloud_backup/cloud_download/templates/cloud_download/aws_login.html
Description:    html template for files
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:               Ryan Breitenfeldt
                       Noah Farris
                       Trevor Surface
                       Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static 'cloud_download/aws.css' %}">
        <meta charset="UTF-8">
    </head>

    <form method="post">
        <div class="box">
        <h1>Amazon Web Services</h1>
        <h2>Login</h1>

        {% csrf_token %}
        {% for field in form.hidden_fields %}
            {{ field }}
        {% endfor %}

        {% for field in form.visible_fields %}
            {{ field }}
        {% endfor %}
        <input type="submit" value="Submit" div class="btn">
        </div> <!-- End Box -->

    </form>
    {% if messages %}
    <ul class="messages">
        {% for message in messages %}
        <script>alert("{{ message }}")</script>
        {% endfor %}
    </ul>
    {% endif %}
</html>
```

project_root/cloud_download/templates/cloud_download/files.html

```
<!--
file page template for cloud backup application

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/templates/cloud_download/files.html
Description:    html template for files
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-Cities
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static 'cloud_download/style.css' %}">
    </head>
        <div class = "outer">
            <div class = "header"><h1>Cloud Download Application</h1></div>
            <div class = "inner">
            <form id="aclass" action="" method="post">
                {% csrf_token %}
                <div class="boxes">
                <input type="checkbox" id = "sel_all" name = "box" onclick="toggle(this)">
                <label for="sel_all"> Select All </label>
                {% for file in files %}
                <input type="checkbox" id = "{{ file }}" value="{{ file }}" name="box">
                <label for="{{ file }}">{{ file.name }} </label>
                {% empty %}
                    <label name="box" for="no_files">No files found... </label>
                {% endfor %}
                </div>
                <input class = "button button1" type="submit" value="Download">
                <a class="button button2" id="back" href="/cloud">Start Over</a>
            </form>
            </div>
        </div>

    <script>
        function toggle(source) {
          checkboxes = document.getElementsByName('box');
          for(var i=0, n=checkboxes.length;i<n;i++) {
            checkboxes[i].checked = source.checked;
          }
        }
```

```
    </script>
</html>
```

project_root/cloud_download/templates/cloud_download/googleIndex.html

```
<!--https://developers.google.com/identity/sign-in/web/server-side-flow -->
<!-- The top of file index.html -->
<html itemscope itemtype="http://schema.org/Article">
<head>
  <!-- BEGIN Pre-requisites -->
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
  </script>
  <script src="https://apis.google.com/js/client:platform.js?onload=start" async defer>
  </script>
  <!-- END Pre-requisites -->

  <!-- Continuing the <head> section -->
    <script>
        function start() {
          gapi.load('auth2', function() {
            auth2 = gapi.auth2.init({
              client_id: '204730000731-c0gs1os80ucalj6mto9c1etmaee70is7.apps.googleusercontent.com',

            });
          });
        }
      </script>
    </head>
    <body>
      <!-- ... -->
<!-- Add where you want your sign-in button to render -->
<!-- Use an image that follows the branding guidelines in a real app -->
<button id="signinButton">Sign in with Google</button>
<script>
  $('#signinButton').click(function() {
    // signInCallback defined in step 6.
    auth2.grantOfflineAccess().then(signInCallback);
  });
</script>
<!-- Last part of BODY element in file index.html -->
<script>
    function signInCallback(authResult) {
      if (authResult['code']) {

        // Hide the sign-in button now that the user is authorized, for example:
        $('#signinButton').attr('style', 'display: none');

        // Send the code to the server
        $.ajax({
          type: 'POST',
          url: 'http://localhost:8000',
          // Always include an `X-Requested-With` header in every AJAX request,
          // to protect against CSRF attacks.
          headers: {
```

```
        'X-Requested-With': 'XMLHttpRequest'
      },
      contentType: 'application/octet-stream; charset=utf-8',
      success: function(result) {
        // Handle or verify the server response.
      },
      processData: false,
      data: authResult['code']
    });
  } else {
    // There was an error.
  }
}
</script>
</body>
</html>
```

project_root/cloud_download/templates/cloud_download/index.html

```
<!--
cloud selection page template for cloud backup application

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/templates/cloud_download/index.html
Description:    html template for index page
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-Cities
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static 'cloud_download/style.css' %}">
        <meta charset="UTF-8">
    </head>

    <body>
        <h1>Cloud Backup Application</h1>
        <h2>Please select a cloud application and click 'Submit'.</h2>
        <div class = "outer2">
        <div class = "inner">
          <div class = "button-group">
            <form action="" method="post">
              {% csrf_token %}
              <div class>
              <select class = "button_index button1" id="platform" name="platform">
                {% for platform in platforms %}
                <option value="{{platform}}">{{platform}}</option>
                {% empty %}
                <option value="empty">error, no platforms found</option>
                {% endfor %}
              </select>
             </div>
             <div>
              <input class = "button button1" type="submit" value="Submit">
            </div>
            </form>
          </div>
      </div>
    </div>
    </body>
</html>
```

project_root/cloud_download/templates/cloud_download/success.html

```
<!--
file page template for cloud backup application

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/templates/cloud_download/files.html
Description:    html template for files
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-Cities
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static 'cloud_download/style.css' %}">
    </head>
    <body>
        <h1>Cloud Backup Application</h1>
        <h3>Downloaded Files:</h3>
        <div class="outer3">
            <div class = "table">
            <ul>
                {% for file in files %}
                <li><a>{{ file.name }}</a></li>
                {% empty %}
                <li><a> No files selected for downloading </a></li>
                {% endfor %}
            </ul>
            </div>
            <a class="button btn" id="back" href="/cloud">Start Over</a>
        </div>
    </body>

</html>
```

project_root/cloud_download/platforms/__init__.py

```
#!/usr/bin/env python3

""" Cloud Platforms Module

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/platforms/__init__.py
Description:    Platform module initializer
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
"""

from . import dropbox_script, gDriveDownloader, Api_keys, aws


__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']
```

project_root/cloud_download/platforms/aws.py

```
import boto3
import json
import os

class aws():
    '''
        self._ec2_client: This is the EC2 Client
        self._s3_client: This is the S3 Client
        self._s3_resource: This is the S3 resource.

        Parameters: access_key_id, access_key, region (default: 'us-west-2')
    '''
    def __init__(self, access_key_id, access_key, region = 'us-west-2'):
        self._ec2_client = boto3.client('ec2', region_name = region,
                        aws_access_key_id=access_key_id,
                        aws_secret_access_key=access_key)
        self._s3_client = boto3.client('s3',region_name=region,
                        aws_access_key_id=access_key_id,
                        aws_secret_access_key=access_key)
        self._s3_resource = boto3.resource('s3',region_name=region,
                        aws_access_key_id=access_key_id,
                        aws_secret_access_key=access_key)

    '''
        Return list of available images as a JSON response.
    '''
    def get_image_list(self):
        images = self._ec2_client.describe_images(Owners=['self'])
        return images

    '''
        Returns a list of all buckets that an AWS Account has access to.
    '''
    def get_buckets(self):
        response = self._s3_client.list_buckets()
        buckets_json = response['Buckets']
        buckets_list = []
        for i in buckets_json:
            buckets_list.append(i['Name'])
        return buckets_list

    '''
        Lists all images within a bucket and returns this as a list of dicts.

        Parameters: bucket_name
    '''
    def list_images_in_bucket(self, bucket_name):
        bucket = self._s3_resource.Bucket(bucket_name)
        files_list = []
```

```
        for my_bucket_object in bucket.objects.all():
            files_list.append({'name': my_bucket_object.key})

        data_set = files_list
        return data_set
    '''
        Exports an image to a bucket, you can access a list of images but if they aren't in a bucket
        you cannot download them.

        Parameters: image_id, role_name, bucket_name, format (default = 'VMDK')
    '''
    def export_to_bucket(self, image_id, role_name, bucket_name, format='VMDK'):
        response = self._ec2_client.export_image(
            Description='string',
            DiskImageFormat=format,
            DryRun=False,
            ImageId=image_id,
            RoleName=role_name,
            S3ExportLocation={
                'S3Bucket': bucket_name
            }
        )
        return response

    '''
        Given a bucket and a file name this will download that file.

        Parameters: bucket_name, file_name
    '''
    def download_image(self, bucket_name, file_name):
        bucket = self._s3_resource.Bucket(bucket_name)
        bucket.download_file(file_name, "/Users/noahfarris/Desktop/downloads" +"/" + os.path.basenam
```

project_root/cloud_download/platforms/class_template.py

```
'''
For Admins:
    This Template class is to help developers add additional cloud platforms to the software
'''

'''
class [Platform_NAME](object):
    def __init__(self):
        ## For attributes use convention [NAME]_[FUNCTION]_[PARAM]
        ## The below attributes are strongly recomended to have within new platform classes
        __[Platform_NAME]__user_download_path       #Consider this to be a private function
        __[Platform_NAME]_get_files_list_result     #Conisder this to be a private function
        __[Platform_NAME]_entries_to_download_list  #Consider this to be a private function
        [Platform_NAME]_format_dict
        [Platform_NAME]_flat_dict

    ##  For Admins - the following three functions are designed based off of the OAuth2 requriements
    ##  If the platform that is being added does not have OAuth2 authentication the following three
    ##  are unneccessay.

    def [Platform_NAME]_authentication_flow(self, request):
        # This function call requires a redirect_uri that may need to need to be specified within th
        # of the platform being used. This class should return the Flow component of the OAuth2 auth
        # will likey need a request.session to help manage state.

    def [Platform_NAMe]_authentication_start(self):
        # This function will initiate the starting parameters for the OAuth2 authentication to use t
        # capability appropriately, after this method is created, this object will need to be instat
        # within the veiws.py script. Additionally the urls.py/urlpattern needs to be updated with t
            path('[Platform_NAME]-auth-start', views.[Platform_NAME].[Platform_NAME]_authentication_
        # When the user selects the new plaform, the Views Index class will need to return a redirec
        # redirect('[Platform_NAME]-auth-start')
        # Finally this method should instatiate a redirect_url to created by the authentication_flow
        # The return will need a django.shortcuts.redirect() to the redirect_url to allow the user t

    def [Platform_NAME]_authentication_finish(self):
        # This fucntion will finalize the OAuth2 authentication. After this method is created, this
        # within the veiws.py script. Additionally the urls.py/urlpattern needs to be updated with t
            path('[Platform_NAME]-auth-finish', views.[Platform_NAME].[Platform_NAME]_authentication
        # The authentication_flow, may require a redirect_uri that includes the full uri to the [Pla
        # This function will catch any errors that arise within the OAuth2 authentication flow.
        # This function will also call all neccessary function to gather files and folders of the ap
        # The returned object will then need to be formated into either a dict specified as:
            {'path':'', 'dirs':[], 'files':[]} in which every dir contains the same dict and dict li
        # Currently the software requires a flat_dict, in which all files are pulled out along with
        # attributes neccessary for downloading. The flact dict will be written as
            {'file':[]} with all the attributes required for downloading files.

    def [Platform_NAME]_get_files_list(self):
```

```
        # This is a generic template for gathering files, use selected platforms api to gather the a
        # Often this may have a list of file entires as a return

    def [Platform_NAME]_format_entries_list(self):
        # This function will take the return of [NAME]_get_files_list
        # And provide the neccessary information to pass to the views context to display the applica

    def [Platform_NAME]_download_selected_entries(self, path):
        # This function will download the list returned by [NAME]_select_entries_to_download
        # Using the [NAME]_download_path to specify the location on the server where the files and f
        # This function should also include a warning that if a file is alread known on a system, it
'''
```

project_root/cloud_download/platforms/dropbox_script.py

```
#!/usr/bin/env python3

''' Dropbox Platform
Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/platforms/dropbox_script.py
Description:    Dropbox cloud platform
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Trevor Surface
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
'''
import os
import dropbox
import json
from django.shortcuts import redirect
from . import Api_keys
import ast
__author__ = "Trevor Surface"

'''
dropbox.oauth.DropboxOAuth2Flow(key, secret, redirect_uri, session, token_type)
dropbox.oauth.DropboxOAuth2Flow(key, secret, redirect_uri, session, token_type).start()
    returns: redirect URL STR
dropbox.oauth.DropboxOAuth2Flow(key, secret, redirect_uri, session, token_type).finish(GET_parameter
    returns: OAuth2FlowResult
    raises: dropbox.oauth.BadRequestException
            dropbox.oauth.BadStateException
            dropbox.oauth.CsrfException
            dropbox.oauth.NotApprovedException
            dropbox.oauth.ProviderException
dropbox.dropbox.Dropbox(access_token)
dropbox.dropbox.Dropbox(access_token).list_folder_result(path)
  returns: dropbox.files.Metadata()
      includes:   files.Metadata.name
                  files.Metadata.path_lower
                  files.Metadata.path_display
                  files.Metadata.parent_shared_folder_id
  raises: dropbox.exceptions.ApiError()
      includes:   request_id
                  error
                  user_message_text
dropbox.dropbox.Dropbox(access_token).files_list_folder_continue(cursor)
  retuns: dropbox.files.Metadata()
  raises: dropbox.exceptions.ApiError()
dropbox.dropbox.Dropbox(access_token).files_download_to_file(download_path, path)
   returns: dropbox.files.FileMetadata()
       includes:   files.FileMetadata.id
```

```
                    files.FileMetadata.client_modified
                    files.FileMetadata.server_modified
                    files.FileMetadata.rev
                    files.FileMetadata.size
                    files.FileMetadata.media_info
                    files.FileMetadata.symlink_info
                    files.FileMetadata.sharing_info
                    files.FileMetadata.is_downloadable
                    files.FileMetadata.export_info
                    files.FileMetadata.property_groups
                    files.FileMetadata.has_explicit_shared_members
                    files.FileMetadata.content_hash
dropbox.dropbox.Dropbox(access_token).files_download_zip_to_file(download_path, path)
    returns: dropbox.files.DownloadZipResult
    raises: dropbox.exceptions.ApiError()
        includes:   dropbox.files.DownloadZipError
'''

class DropBox(object):
    def __init__(self):
        '''
        __init__
        Attributes to modify:
        self.__dropbox_api_key, self.__dropbox_api_secret
            Admin needs to create an app via the dropbox app console: https://dropbox.com/developer
            Once the app has been created, Admin will get both a KEY and a SECRET, both of which
            should be added to an API_Keys.py file, with parameter names Dropbox_Api_key and
            Dropbox_Api_secret. Make sure to include the file in the .gitignore.
        self.__dropbox_download_path
            Admin needs to specify a location on the server that will store the downloaded files,
            additionally, for larger applications, the user may be asked to specify a location.
        '''
        self.dbx = None
        '''
            This attribute will contain the dropbox object after a users has been authenticated
        '''
        self.__dropbox_api_key = Api_keys.Dropbox_Api_key
        '''
        See __init__, Attributes to modify
        '''
        self.__dropbox_api_secret = Api_keys.Dropbox_Api_secret
        '''
        See __init__, Attributes to modify
        '''
        self.__dropbox_authentication_oauth_result = ""
        '''
            This attribute will contain the oath information, which includes the access token
            See the dropbox function definition at the top of the screen
        '''
        self.__dropbox_get_files_return = ""
        '''
```

```
        This attribute will contain the result returned by the dropbox.get_file_list_folder()
        api call, defined in the above api definitions
    '''
    self.__dropbox_get_files_list_result = []
    '''
        This attribute will contain a list of results with information from the previous attribu
        accessing the .entries parameter of the return
    '''
    self.__dropbox_download_path = os.environ['HOME'] + "/Downloads"
    '''
        See __init__, Attributes to modify
    '''
    self.dropbox_format_dict = {'path': '', 'dirs':[], 'files':[]}
    '''
        This attribute holds a dict for each folder, containing a variable list of folders in ea
        and the included files within a given folder
    '''
    self.dropbox_flat_dict = {'files':[]}
    '''
        This attribute holds a flattened dict that will only show users files, not including the
        of the neccessary files.
    '''
    self.__dropbox_files_paths = []
    '''
        This attribute stores the paths of the files returned from the dropbox.get_file_list_fol
    '''

def dropbox_auth_flow(self, request):
    '''
        This function generates the flow object that needs to be passed through
        the authentication_start() method and the authentication_finish() method

    For Admin's:
        The redirect_uri parameter needs to be set in both the urls.py/urlpatterns as
        a path, which calls the dropbox_authetication_finish function. Additionally
        in the app console for dropbox (https://dropbox.com/developer) the same redirect_uri
        needs to be specified for the generated application.
    '''
    redirect_uri = "http://localhost:8000/cloud/dropbox-auth-finish"
    return dropbox.oauth.DropboxOAuth2Flow(self.__dropbox_api_key, self.__dropbox_api_secret, re

def dropbox_authentication_start(self, web_app_session):
    '''
        This function generates the redirect for the application users to login to their
        dropbox accounts
    '''
    authorize_url = self.dropbox_auth_flow(web_app_session.session).start()
    return redirect(authorize_url)

def dropbox_authentication_finish(self, request):
    '''
```

```
        This function finalizes the authentication with the application user

    For Admins:
        The exceptions for Bad State, needs to be updated to the path that initialized the
        authentication. This needs to be updated in the urls.py/urlpatterns, with a path that
        calls the dropbox_authentication_start method.

        Additionally the Not Approved Exception return will need to be updated so that the appli
        user is redirected to the home page of the webiste.
    '''
    try:
        self.__dropbox_authentication_oauth_result = self.dropbox_auth_flow(request.session).fin
    except dropbox.oauth.BadRequestException as e:
        raise e
    except dropbox.oauth.BadStateException as e:
        return redirect("http://localhost:8000/cloud/dropbox-auth-start/")
    except dropbox.oauth.CsrfException as e:
        return HttpResponseForbidden
    except dropbox.oauth.NotApprovedException as e:
        flash('Not approved?  Why not?')
        return redirect("http://localhost:8000/cloud")
    except dropbox.oauth.ProviderException as e:
        logger.log("Auth error: %s" % (e,))
        raise e
    self.dbx = dropbox.dropbox.Dropbox(self.__dropbox_authentication_oauth_result.access_token)
    ''' The above function call uses the oauth_return to generate the dropbox object'''
    self.dropbox_get_files_list()
    ''' The above function uses the dropbox api's to gather the list of folders and files the au
        has in their dropbox account, for further details see the function definition below'''
    self.dropbox_format_entries_recur(self.dropbox_format_dict, 1, self.__dropbox_files_paths)
    ''' The above funtion generates a dict, or json, depending upon the use of the #json.dumps c
        function defined below. The admin can change the views page to either display the json o
        #self.dropbox_format_dict = json.dumps(self.dropbox_format_dict)
    self.dropbox_dict_flatten_recur(self.dropbox_format_dict)
    ''' The above function flattens the dict to only show the files that are available on the ap
        dropbox account, this can be removed if the Admin would like to show the directories WAR
        is only set up to show a flat_dict object, and will need further editing to enable the f
    return redirect('/cloud/files')
    ''' The final part of the function will redirect the user to the files view.py page, where t
        be displayed for them to choose which to download to the server'''

def dropbox_get_files_list(self):
    '''
        This function calls the files_list_to_folder() dropbox function, gathering the list of a
        files and folders. The application will then generate two lists in order to create a pro
        in the cloud/files/ view.
    '''
    try:
        self.__dropbox_get_files_list_return = self.dbx.files_list_folder("", recursive=True)
    except dropbox.exception.ApiError as e:
        raise e
```

```
    for dropbox_files in self.__dropbox_get_files_list_return.entries:
        self.__dropbox_get_files_list_result.append(dropbox_files)
        self.__dropbox_files_paths.append(dropbox_files.path_lower)

def dropbox_format_entries_recur(self, build_dict, level, file_list):
    '''
        This function is a recursive algorithm designed to create a multi layered dict that will
        root folder, and the subsequent folders that a application user may have available in th
        WARNING: For the Admins, the current functionaliry is based on the fact that a user does
        in their directory names, however, if they do the generated dict will be invalid and not
        files and folders, this would need to be changed to allow file/folder detection.
    '''
    for file_paths in file_list:
        if file_paths.count('/') == level and build_dict['path'] in file_paths:
            dict = {'path': file_paths}
            if '.' in file_paths:
                dict['name'] = file_paths.split('/')[len(file_paths.split('/')) - 1]
                if dict not in build_dict['files']:
                    build_dict['files'].append(dict)
            else:
                dict['dirs'] = []
                dict['files'] =[]
                self.dropbox_format_entries_recur(dict, level + 1, file_list)
                if dict not in build_dict['dirs']:
                    build_dict['dirs'].append(dict)

def dropbox_dict_flatten_recur(self, dir):
    '''
        This function uses a recursive algorithm to strip off directories from a application use
        system, only displaying all of their current files.
    '''
    for files in dir['files']:
        self.dropbox_flat_dict['files'].append(files)
    for dirs in dir['dirs']:
        self.dropbox_dict_flatten_recur(dirs)

def dropbox_download_selected_entries(self, download_list):
    '''
        This function is called after an application user has selected the files they wish to do

    For Admins:
        This function works off the premise that users do not put a '.' in their folder names, t
        the case, in which a change would need to be made to allow file/folder detection. Additi
        the software will not detect if there is a currently named file in the system with same
        file. In which case the download will be overwritten by the new download.
    '''
    for files in download_list:
        if '.' in files['path']:
            try:
                self.dbx.files_download_to_file(self.__dropbox_download_path + '/' + files['name
            except dropbox.exceptions.ApiError as e:
```

```
            raise e
    else:
        try:
            self.dbx.files_download_zip_to_file(self.__dropbox_download_path + '/' + files['
        except dropbox.exceptions.ApiError as e:
            raise e
```

project_root/cloud_download/platforms/gDriveDownloader.py

```
#!/usr/bin/env python3

""" Google Drive Cloud Platform

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/platforms/gDriveDownloader.py
Description:    Google Drive Cloud Platform
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
"""

from __future__ import print_function
import pickle
import os.path
import os
import io
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload
from google_auth_oauthlib.flow import InstalledAppFlow
from google_auth_oauthlib.flow import Flow
from google.auth.transport.requests import Request
import google.oauth2.credentials
from google.oauth2.credentials import Credentials
from django.shortcuts import render, redirect

__author__ = 'Ryan Breitenfeldt'

#googleapiclient.discovery.build
#googleapiclient.discovery.build.files().list()
#googleapiclient.http.MediaIoBaseDownload
#google_auth_oauthlib.flow.InstalledAppFlow
#google.auth.transport.requests.Request


class GDriveDownloader():
    def __init__(self,creds=None,service=None):
        self.GDriveDownloader_creds = creds #stores the credetials from google
        self.GDriveDownloader_SCOPES = ['https://www.googleapis.com/auth/drive'] # the scope of what
        self.GDriveDownloader_service = service # the engine for googled api
        self.GDriveDownloader_file_List = None # holds all the meta data and file info from Drive
        self.GDriveDownloader_json = {'files':[]} # the json to be used by the ui
        self.GDriveDownloader_files_to_download = [] # the list of files to download

    #this functoin is for authenticting as a standalone class
    #use the redirect functions below for authentication in Django
```

```
    def GDriveDownloader_authentication(self):
        # If there are no (valid) credentials available, let the user log in.
        if not self.GDriveDownloader_creds or not self.GDriveDownloader_creds.valid:
            if self.GDriveDownloader_creds and self.GDriveDownloader_creds.expired and self.GDriveDo
                creds.refresh(Request())
            else:
                flow = InstalledAppFlow.from_client_secrets_file('client_secret_204730000731-c0gs1os
                self.GDriveDownloader_creds = flow.run_local_server(port=0)

    # saves the credencials to a file
    def GDriveDownloader_save_Token(self):
        with open('token.pickle', 'wb') as token:
            pickle.dump(self.GDriveDownloader_creds, token)

    # load creds from a file
    def GDriveDownloader_load_Token(self):
        if os.path.exists('token.pickle'):
            with open('token.pickle', 'rb') as token:
                creds = pickle.load(token)

    # creates the google drive service from the credentials
    def GDriveDownloader_build_Service(self):
        self.GDriveDownloader_service = build('drive', 'v3', credentials=self.GDriveDownloader_creds

# loops over GDriveDownloader_files_to_download and downloads each file to the working directory
    def GDriveDownloader__download_File(self):
        os.chdir("/Users/noahfarris/Desktop/downloads")
        for file_id in self.GDriveDownloader_files_to_download:
            request = self.GDriveDownloader_service.files().get_media(fileId=file_id["id"]) # reques
            fh = io.BytesIO()
            downloader = MediaIoBaseDownload(fh, request) # makes the downloader for the file
            done = False
            while done is False:
                status, done = downloader.next_chunk()
                print("Download {}%".format(int(status.progress() * 100)))
            f = open(file_id["name"], 'wb')
            f.write(fh.getvalue())
        os.chdir('/Users/noahfarris/Desktop/CAPSTONE_FINAL/git/cloud-backup/cloud_backup')
# makes a query to google drive to get the files. it filters out folders and google propriatary file
    def GDriveDownloader_get_Files(self):
        page_token = None
        self.GDriveDownloader_json['files'].clear() # clears the list of files to prevent duplicatio
        while True:
            self.GDriveDownloader_file_List = self.GDriveDownloader_service.files().list(q="mimeType
            for file in self.GDriveDownloader_file_List["files"]:
                self.GDriveDownloader_json['files'].append({"name": file["name"],"id": file["id"]})
            page_token = self.GDriveDownloader_file_List.get('nextPageToken', None)
            if page_token is None:
                break
```

```
# takes dictionary as input in this formate: {"name": "file name", "id": " file id"}
    def GDriveDownloader_add_file_to_download(self,fileId):
        self.GDriveDownloader_files_to_download.append(fileId)

    #creates the autheration flow for the redirect
    def GDriveDownloaded_authentication_flow(self):
        redirect_uri = "http://localhost:8000/cloud/google-auth-finish/"
        return Flow.from_client_secrets_file(
            'client_secret_204730000731-c0gs1os80ucalj6mto9c1etmaee70is7.apps.googleusercontent.com.
            scopes=self.GDriveDownloader_SCOPES,
            redirect_uri=redirect_uri)

    # redirects to the autheration url
    def GDriveDownloaded_authentication_start(self, request):
        auth_uri = self.GDriveDownloaded_authentication_flow().authorization_url()[0]
        return redirect(auth_uri[:-20])

    # takes a json returned from google and turns it into a credential object
    # it builds the service and gets the list of files.
    def GDriveDownloaded_authentication_finish(self, request):
        try:
            self.GDriveDownloader_creds = self.GDriveDownloaded_authentication_flow().fetch_token(co
            #print(self.GDriveDownloader_creds)
            self.GDriveDownloader_creds = Credentials(self.GDriveDownloader_creds['access_token'])
            self.GDriveDownloader_build_Service()
            self.GDriveDownloader_get_Files()
        except Exception as e:
            raise e
        return redirect('/cloud/files')
```