Python Assignment

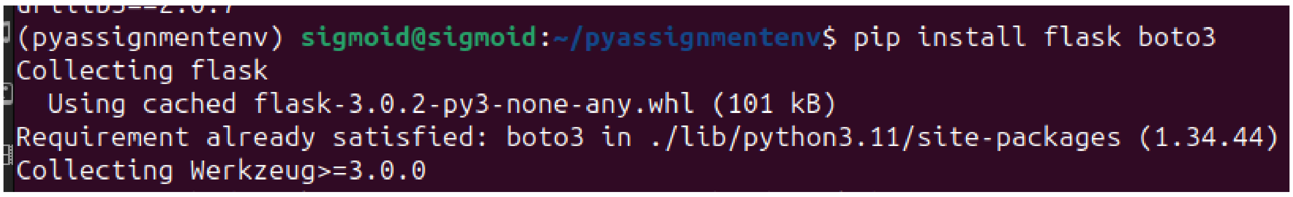
Question:
Implement s3 file manager using any python web framework(flask/django/...etc).
functions :
1. List content of s3.
2. Create/Delete folder + bucket .
3. Upload files to s3 + delete file from s3.
4. Copy/Move file withing s3.

Note:
1. Make sure your code is readable
2. Make sure your app is working properly
3. Need basic UI from which we can access app

Solution:
We will achieve this using Flask Framework.

Step 1:
First we need to install Flask and Boto3 using command
pip install flask boto3

or
pip install flask
pip install boto3



```
(pyassignmentenv) sigmoid@sigmoid:~/pyassignmentenv$ pip install flask boto3
Collecting flask
  Using cached flask-3.0.2-py3-none-any.whl (101 kB)
Requirement already satisfied: boto3 in ./lib/python3.11/site-packages (1.34.44)
Collecting Werkzeug>=3.0.0
```

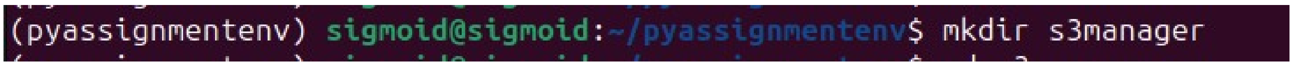
Flask:- Flask is a lightweight web application framework for Python. It is designed to be simple, easy to use, and flexible, allowing developers to quickly build web applications with minimal boilerplate code.

Boto3:- Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python. It allows Python developers to write software that makes use of services like Amazon S3, EC2, DynamoDB, and many others offered by AWS.

Step 2 :
create a project folder in which we can create our required files.

mkdir <projectname>

eg: mkdir s3manager



```
(pyassignmentenv) sigmoid@sigmoid:~/pyassignmentenv$ mkdir s3manager
```

Enter into the project folder using cd command
cd s3manager

Python Assignment

Step 3:
Create an app.py file or any python file as per your requirement this file will be our application.

```
touch app.py
```

Step 4:
open that file in any editor or notepad (I have used VS Code).
And import flask and boto3 in that file..
And also import required modules like render_template,request,url_for and redirect.
And also import os.

Reasons to import all these modules:
render_template : This module is used to render HTML templates. In our application, we use HTML templates to generate the content of web pages dynamically. For example, the `render_template` function is used to render the `index.html` template, which displays the contents of the S3 bucket.

request : This module provides access to incoming request data. In our application, we use `request` to retrieve form data submitted by users. For instance, when a user submits a form to create a new folder or upload a file, we use `request` to extract the relevant data (e.g., folder name, uploaded file).

redirect : This module is used to perform HTTP redirects. After processing a request, we often want to redirect the user to another route or URL. For example, after creating a new folder or uploading a file, we redirect the user back to the index route to display the updated contents of the S3 bucket.This module is used to perform HTTP redirects. After processing a request, we often want to redirect the user to another route or URL. For example, after creating a new folder or uploading a file, we redirect the user back to the index route to display the updated contents of the S3 bucket.

url_for :   This module is used to generate URLs for routes defined in the application. Instead of hardcoding URLs in the templates or views, we use `url_for` to generate URLs dynamically based on the route names. This makes our application more flexible and easier to maintain, especially if route URLs change in the future.

Os : We are importing os because the `os` module is imported in the provided code because it provides a portable way of using operating system-dependent functionality. In this specific Flask application, the `os` module is used to perform path manipulations for file operations.

Step 5:
Add this line our python file (app.py) :

```
app = Flask(__name__)
```

This line creates a Flask application instance.

Python Assignment

Step 6:

```
AWS_ACCESS_KEY_ID = 'your_access_key_id'
AWS_SECRET_ACCESS_KEY = 'your_secret_access_key'
S3_BUCKET_NAME = 'your_bucket_name'
```

Add this line in our app.py file to define the AWS access credentials (Access Key ID and Secret Access Key) and the AWS region. These credentials are used to authenticate and authorize requests to AWS services.

Step 7:

```
s3 = boto3.client('s3',
                  aws_access_key_id=AWS_ACCESS_KEY_ID,
                  aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
                  region_name=AWS_REGION)
```

This line creates an S3 client object using the `boto3.client()` method. It uses the provided AWS access credentials and region to authenticate requests and interact with the Amazon S3 service.

Step 8:

```
BUCKET_NAME = 's3access-file'
```

This line defines the name of the S3 bucket that will be used by the application.

Step:9

```
if __name__ == '__main__':
    app.run(debug=True)
```

Add this line at the end of the file.This block of code runs the Flask application when the script is executed directly (`__name__ == '__main__'`). It runs the application in debug mode (`debug=True`), which enables helpful debugging features and automatically reloads the application when code changes are detected.

Now all the configuration has been completed lets create the required functions:

Note: Write all the functions before step 9.

Implement the required functions for managing S3 files, including listing contents, creating/deleting folders and buckets, uploading/deleting files, and copying/moving files within S3.

Python Assignment

Function for listing content

```
@app.route('/')
def index():
    # List contents of the S3 bucket
    response = s3.list_objects(Bucket=BUCKET_NAME)
    files = []
    folders = {}
    if 'Contents' in response:
        for obj in response['Contents']:
            if obj['Key'].endswith('/'):
                folder_name = obj['Key']
                folders[folder_name] = []
            else:
                file_name = obj['Key']
                folder_name = '/'.join(file_name.split('/')[:-1])
                folders[folder_name].append(file_name)
                files.append(file_name)  # Collect all file names
    return render_template('index.html', folders=folders, files=files)
```

This code defines a Flask route for the root URL ("/"), which renders the index page of a web application. Within the route function:

1. It sends a request to the Amazon S3 service to list the objects (files and folders) in a specific S3 bucket (`BUCKET_NAME`).
2. It initializes empty lists (`files`) and an empty dictionary (`folders`) to store the names of files and folders retrieved from the S3 bucket, respectively.
3. It checks if the response from the S3 service contains any objects. If it does, it iterates through each object, determining whether it's a file or a folder.
4. If it's a folder, it adds it to the `folders` dictionary with an empty list as its value.
5. If it's a file, it extracts the file name and its parent folder's name, adding the file name to the list of files and appending the file name to the list of files within its parent folder in the `folders` dictionary.
6. Finally, it renders the `index.html` template, passing the `folders` dictionary and the `files` list as arguments. This data is used to dynamically generate the content of the index page, displaying the folders and files stored in the S3 bucket.

Lets understand this code line by line:

```
@app.route('/')
```

```
def index():
```

This decorator (`@app.route('/')`) specifies that the following function (`index()`) should be executed when the root URL (i.e., `'/'`) of the web application is accessed.

Python Assignment

```
  response = s3.list_objects(Bucket=BUCKET_NAME)
```

This line sends a request to the Amazon S3 service to list the objects (files and folders) in the specified S3 bucket (`BUCKET_NAME`). The response is stored in the `response` variable.

```
    files = []
```

```
    folders = {}
```

These lines initialize empty lists (`files`) and an empty dictionary (`folders`) to store the names of files and folders retrieved from the S3 bucket, respectively.

```
    if 'Contents' in response:
```

This line checks if the response from the S3 service contains the key `'Contents'`. If it does, it means that there are objects (files or folders) in the bucket.

```
        for obj in response['Contents']:
```

This loop iterates through each object (file or folder) listed in the `response['Contents']`.

```
            if obj['Key'].endswith('/'):
```

This condition checks if the object key (i.e., its name) ends with a forward slash (`'/'`). If it does, it indicates that the object is a folder.

```
                folder_name = obj['Key']
```

```
                folders[folder_name] = []
```

If the object is a folder, its name is stored in the `folder_name` variable, and an empty list is assigned to it in the `folders` dictionary. This creates an entry for the folder where its contents (files) can be stored later.

```
            else:
```

If the object key does not end with a forward slash, it indicates that the object is a file.

```
                file_name = obj['Key']
```

```
                folder_name = '/'.join(file_name.split('/')[:-1])
```

```
                folders[folder_name].append(file_name)
```

```
                files.append(file_name)  # Collect all file names
```

For files, the code extracts the file name from the object key and determines the parent folder's name. It then adds the file name to the list of files (`files`) and appends the file name to the list of files within its parent folder in the `folders` dictionary.

```
    return render_template('index.html', folders=folders, files=files)
```

Finally, this line renders the `index.html` template, passing the `folders` dictionary and the `files` list as arguments. These variables contain information about the folders and files in the S3 bucket, which will be used to dynamically generate the content of the index page.

Python Assignment

We have written the code to list the contents. Now will proceed with the operations such as creation and deletion of folders(bucket), upload and delete files and copy and move files.

Function for creating folder:

```
@app.route('/create-folder', methods=['POST'])
def create_folder():
    folder_name = request.form['folder-name']
    if folder_name:
        # Create folder in S3 bucket
        s3.put_object(Bucket=BUCKET_NAME, Key=(folder_name + '/'))
    return redirect(url_for('index'))
```

This route allows users to create a new folder in the S3 bucket by submitting a form with the desired folder name. After the folder is created, the user is redirected to the index page to see the updated contents of the S3 bucket.

```
@app.route('/create-folder', methods=['POST'])
```

This line is a decorator that specifies the URL endpoint (`'/create-folder'`) for the route. It also defines that this route will only respond to POST requests (`methods=['POST']`). This means that this route will handle form submissions where the HTTP method is POST.

```
def create_folder():
```

This line defines the function `create_folder()` that will be executed when a POST request is made to the '/create-folder' endpoint.

```
folder_name = request.form['folder-name']
```

This line retrieves the value of the form field named `'folder-name'` from the request object. It expects that the form submitted to this route contains a field named `'folder-name'`, which represents the name of the folder to be created.

```
if folder_name:if folder_name:
```

This line checks if a folder name was provided. It evaluates to `True` if `folder_name` is not empty, indicating that the user has submitted a non-empty folder name.

```
s3.put_object(Bucket=BUCKET_NAME, Key=(folder_name + '/'))
```

This line sends a request to the Amazon S3 service to create a new object (folder) in the specified S3 bucket (`BUCKET_NAME`). The `put_object` method is used to create a new object, and the `Key` parameter specifies the key (or path) of the new object. Here, `folder_name + '/'` represents the path of the new folder with a trailing forward slash to indicate it's a folder.

```
return redirect(url_for('index'))
```

Python Assignment

After the folder creation is complete, this line redirects the user to the index route (`index()`). The `redirect()` function is used to perform an HTTP redirect, and `url_for('index')` generates the URL for the index route based on its name. So, the user is redirected back to the index page, where they can see the updated contents of the S3 buckets.

Function for deleting the folder:

```
@app.route('/delete-folder', methods=['POST'])
def delete_folder():
    folder_name = request.form['folder-name']
    if folder_name:
        # Delete folder and its contents from S3 bucket
        s3.delete_object(Bucket=BUCKET_NAME, Key=(folder_name))
    return redirect(url_for('index'))
```

This code defines a Flask route for deleting a folder from the S3 bucket. It expects the folder name to be submitted via a POST request to the '/delete-folder' endpoint. Once the folder name is received, it sends a request to the Amazon S3 service to delete the specified folder and its contents from the bucket. After successful deletion, it redirects the user back to the index page to view the updated contents of the S3 bucket.

```
@app.route('/delete-folder', methods=['POST'])
```

This line is a decorator that specifies the URL endpoint (`'/delete-folder'`) for the route. It also defines that this route will only respond to POST requests (`methods=['POST']`).

```
def delete_folder():
```

This line defines the function `delete_folder()` that will be executed when a POST request is made to the '/delete-folder' endpoint.

```
folder_name = request.form['folder-name']
```

This line retrieves the value of the form field named `'folder-name'` from the request object. It expects that the form submitted to this route contains a field named `'folder-name'`, which represents the name of the folder to be deleted.

```
if folder_name:
```

This line checks if a folder name was provided. It evaluates to `True` if `folder_name` is not empty, indicating that the user has submitted a non-empty folder name.

Python Assignment

s3.delete_object(Bucket=BUCKET_NAME, Key=(folder_name))

This line sends a request to the Amazon S3 service to delete the specified folder (`folder_name`) and its contents from the S3 bucket (`BUCKET_NAME`). The `delete_object` method is used to delete the folder.

return redirect(url_for('index'))

After the folder deletion is complete, this line redirects the user to the index route (`index()`). The `redirect()` function is used to perform an HTTP redirect, and `url_for('index')` generates the URL for the index route based on its name. So, the user is redirected back to the index page, where they can view the updated contents of the S3 bucket.

Function for deleting files.

```
 @app.route('/delete-file', methods=['POST'])
def delete_file():
    file_name = request.form['file-name']
    if file_name:
        # Delete file from S3 bucket
        s3.delete_object(Bucket=BUCKET_NAME, Key=file_name)
    return redirect(url_for('index'))
```

This code defines a Flask route for deleting a file from the S3 bucket. It expects the file name to be submitted via a POST request to the '/delete-file' endpoint. Once the file name is received, it sends a request to the Amazon S3 service to delete the specified file from the bucket. After successful deletion, it redirects the user back to the index page to view the updated contents of the S3 bucket.

@app.route('/delete-file', methods=['POST'])

This line is a decorator that specifies the URL endpoint (`'/delete-file'`) for the route. It also defines that this route will only respond to POST requests (`methods=['POST']`).

def delete_file():

This line defines the function `delete_file()` that will be executed when a POST request is made to the '/delete-file' endpoint.

file_name = request.form['file-name']

This line retrieves the value of the form field named `'file-name'` from the request object. It expects that the form submitted to this route contains a field named `'file-name'`, which represents the name of the file to be deleted.

Python Assignment

if file_name:

This line checks if a file name was provided. It evaluates to `True` if `file_name` is not empty, indicating that the user has submitted a non-empty file name.

s3.delete_object(Bucket=BUCKET_NAME, Key=file_name)

This line sends a request to the Amazon S3 service to delete the specified file (`file_name`) from the S3 bucket (`BUCKET_NAME`). The `delete_object` method is used to delete the file.

return redirect(url_for('index'))

After the file deletion is complete, this line redirects the user to the index route (`index()`). The `redirect()` function is used to perform an HTTP redirect, and `url_for('index')` generates the URL for the index route based on its name. So, the user is redirected back to the index page, where they can view the updated contents of the S3 bucket.

Function to upload file:

```
@app.route('/upload-file', methods=['POST'])
def upload_file():
    file = request.files['file']
    folder = request.form['folder']
    if file:
        # Upload file to S3 bucket in the selected folder
        s3.upload_fileobj(file, BUCKET_NAME, folder + '/' + file.filename)
    return redirect(url_for('index'))
```

This code defines a Flask route for uploading a file to the S3 bucket. It expects a file to be uploaded along with the name of the folder where the file should be placed, submitted via a POST request to the '/upload-file' endpoint. Once the file and folder information are received, it uploads the file to the specified folder in the S3 bucket. After successful upload, it redirects the user back to the index page to view the updated contents of the S3 bucket.

Python Assignment

Function to copy file within s3:

```
@app.route('/copy-file', methods=['POST'])
def copy_file():
    source_file = request.form['source-file']
    destination_folder = request.form['destination-folder']
    if source_file and destination_folder:
        # Copy file within S3 bucket
        s3.copy_object(Bucket=BUCKET_NAME,
CopySource=f"{BUCKET_NAME}/{source_file}", Key=(destination_folder + '/' +
os.path.basename(source_file)))
    return redirect(url_for('index'))
```

This code defines a Flask route for copying a file within the S3 bucket. It expects the name of the source file and the destination folder where the file should be copied to, submitted via a POST request to the '/copy-file' endpoint. Once the source file and destination folder information are received, it sends a request to the Amazon S3 service to copy the specified file to the specified destination folder within the same bucket. After successful copy, it redirects the user back to the index page to view the updated contents of the S3 bucket.

Function to move file within s3:

```
    @app.route('/move-file', methods=['POST'])
def move_file():
    source_file = request.form['source-file']
    destination_folder = request.form['destination-folder']
    if source_file and destination_folder:
        # Move file within S3 bucket (copy then delete)
        s3.copy_object(Bucket=BUCKET_NAME,
CopySource=f"{BUCKET_NAME}/{source_file}", Key=(destination_folder + '/' +
os.path.basename(source_file)))
        s3.delete_object(Bucket=BUCKET_NAME, Key=source_file)
    return redirect(url_for('index'))
```

This code defines a Flask route for moving a file within the S3 bucket. It expects the name of the source file and the destination folder where the file should be moved to, submitted via a POST request to the '/move-file' endpoint. Once the source file and destination folder information are received, it first copies the specified file to the specified destination folder within the same bucket and then deletes the original file. After successfully moving the file, it redirects the user back to the index page to view the updated contents of the S3 bucket.

After doing all the configuration and creating all the functions our app.py file should be look like this:

```
from flask import Flask, render_template, request, redirect, url_for

import boto3

import os


app = Flask(__name__)


# Configure AWS credentials

AWS_ACCESS_KEY_ID = 'AKIAQT5APJTNRFRAYNT3'

AWS_SECRET_ACCESS_KEY = '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'

AWS_REGION = 'ap-south-1'


# Create an S3 client

s3 = boto3.client('s3',

                  aws_access_key_id=AWS_ACCESS_KEY_ID,

                  aws_secret_access_key=AWS_SECRET_ACCESS_KEY,

                  region_name=AWS_REGION)


# Define bucket name

BUCKET_NAME = 's3access-file'


@app.route('/')

def index():

    # List contents of the S3 bucket

    response = s3.list_objects(Bucket=BUCKET_NAME)

    files = []

    folders = {}

    if 'Contents' in response:
```

Python Assignment

```
        for obj in response['Contents']:
            if obj['Key'].endswith('/'):
                folder_name = obj['Key']
                folders[folder_name] = []
            else:
                file_name = obj['Key']
                folder_name = '/'.join(file_name.split('/')[:-1])
                folders[folder_name].append(file_name)
                files.append(file_name)  # Collect all file names
    return render_template('index.html', folders=folders, files=files)

@app.route('/create-folder', methods=['POST'])
def create_folder():
    folder_name = request.form['folder-name']
    if folder_name:
        # Create folder in S3 bucket
        s3.put_object(Bucket=BUCKET_NAME, Key=(folder_name + '/'))
    return redirect(url_for('index'))



@app.route('/delete-folder', methods=['POST'])
def delete_folder():
    folder_name = request.form['folder-name']
    if folder_name:
        # Delete folder and its contents from S3 bucket
        s3.delete_object(Bucket=BUCKET_NAME, Key=(folder_name))
    return redirect(url_for('index'))

@app.route('/delete-file', methods=['POST'])
def delete_file():
```

Python Assignment

```
    file_name = request.form['file-name']
    if file_name:
        # Delete file from S3 bucket
        s3.delete_object(Bucket=BUCKET_NAME, Key=file_name)
    return redirect(url_for('index'))

@app.route('/upload-file', methods=['POST'])
def upload_file():
    file = request.files['file']
    folder = request.form['folder']
    if file:
        # Upload file to S3 bucket in the selected folder
        s3.upload_fileobj(file, BUCKET_NAME, folder + '/' + file.filename)
    return redirect(url_for('index'))

@app.route('/copy-file', methods=['POST'])
def copy_file():
    source_file = request.form['source-file']
    destination_folder = request.form['destination-folder']
    if source_file and destination_folder:
        # Copy file within S3 bucket
        s3.copy_object(Bucket=BUCKET_NAME, 
CopySource=f"{BUCKET_NAME}/{source_file}", Key=(destination_folder + '/' + 
os.path.basename(source_file)))
    return redirect(url_for('index'))

@app.route('/move-file', methods=['POST'])
def move_file():
    source_file = request.form['source-file']
    destination_folder = request.form['destination-folder']
    if source_file and destination_folder:
```

Python Assignment

```
        # Move file within S3 bucket (copy then delete)
        s3.copy_object(Bucket=BUCKET_NAME, CopySource=f"{BUCKET_NAME}/{source_file}", Key=(destination_folder + '/' + os.path.basename(source_file)))
        s3.delete_object(Bucket=BUCKET_NAME, Key=source_file)
    return redirect(url_for('index'))


if __name__ == '__main__':
    app.run(debug=True)
```

After creating the functions create one templates folder in project folder (s3manager in our case).

```
Mkdir templates
```

Now create a index.html file in templates and write code to list contents and other operations.

```
Cd templates
touch index.html
```

write html code in index.html file

After that create one static folder to store css file to style the UI.

```
Mkdir static
```

create one css file style.css

```
cd static
touch style.css
```