Python Assignment **TAS425**

Question 1
(5 points)
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 within 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

The S3 File Manager is a web application built using the Flask web framework

Technologies Used:

- Python: Backend logic and application flow.
- Flask: Web framework for building the API and UI.
- Boto3: AWS SDK for Python to interact with S3.
- HTML and CSS: Basic UI for interacting with the application.
- JavaScript: Handling dynamic UI actions like listing files.

Application Structure:

```
S3FileManager/
├── app.py          # Flask application
├── template/
  └── index.html   # HTML template
```

Flask Application **(app.py)** :

The application is initialized using Flask and configured to use the **Boto3** library to connect to S3.

```
from flask import Flask, request, jsonify, render_template
import boto3
import botocore.exceptions

app = Flask(__name__, template_folder="template")
s3 = boto3.client("s3")


@app.route("/")
def index():
        """Render the main page."""
        return render_template("index.html")


@app.route("/list")
def list_files():
        """List all files in a specified S3 bucket."""
        bucket_name = request.args.get("bucket_name")

        if not bucket_name:
        return jsonify({"error": "Bucket name is required"}), 400

        try:
        response = s3.list_objects_v2(Bucket=bucket_name)
        objects = response.get("Contents", [])

        if not objects:
        return jsonify({"message": "No files found in the bucket"}), 404

        files = [obj["Key"] for obj in objects]
        return jsonify(files)

        except botocore.exceptions.NoCredentialsError:
        return jsonify({"error": "AWS credentials not found"}), 401

        except botocore.exceptions.ParamValidationError:
        return jsonify({"error": "Invalid parameters provided"}), 400
        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500

@app.route("/create_folder", methods=["POST"])
def create_folder():
        """Create a folder in an S3 bucket."""
        try:
        bucket_name = request.form.get("bucket_name")
        folder_name = request.form.get("folder_name")

        if not bucket_name or not folder_name:
        return jsonify({"error": "Bucket name and folder name are required"}), 400

        # Ensure folder name ends with a slash to be treated as a folder
        if not folder_name.endswith("/"):
        folder_name += "/"

        # Upload an empty file to create the folder
        s3.put_object(Bucket=bucket_name, Key=folder_name)
        return jsonify({"message": f"Folder '{folder_name}' created successfully in bucket
'{bucket_name}'"})

        except botocore.exceptions.NoCredentialsError:
        return jsonify({"error": "AWS credentials not found"}), 401

        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500



@app.route("/delete_folder", methods=["POST"])
def delete_folder():
        """Delete a folder and all its contents from an S3 bucket."""
        try:
        bucket_name = request.form.get("bucket_name")
        folder_name = request.form.get("folder_name")

        if not bucket_name or not folder_name:
        return jsonify({"error": "Bucket name and folder name are required"}), 400
        # Ensure the folder name ends with a slash
        if not folder_name.endswith("/"):
        folder_name += "/"

        # List all objects within the folder
        response = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_name)
        objects = response.get("Contents", [])

        if not objects:
        return jsonify({"message": f"Folder '{folder_name}' not found or already empty in bucket
'{bucket_name}'"}), 404

        # Collect all object keys to delete
        keys = [{"Key": obj["Key"]} for obj in objects]

        # Delete all the objects within the folder
        delete_response = s3.delete_objects(
        Bucket=bucket_name,
        Delete={"Objects": keys}
        )

        return jsonify({"message": f"Folder '{folder_name}' and its contents deleted successfully from
'{bucket_name}'"})

        except botocore.exceptions.NoCredentialsError:
        return jsonify({"error": "AWS credentials not found"}), 401

        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500


@app.route("/upload", methods=["POST"])
def upload_file():
        """Upload a file to a specified S3 bucket."""
        try:
        if "file" not in request.files:
        return jsonify({"error": "No file provided"}), 400

        bucket_name = request.form.get("bucket_name")
        file = request.files["file"]
        if not bucket_name:
        return jsonify({"error": "Bucket name is required"}), 400

        if file.filename == "":
        return jsonify({"error": "Empty filename"}), 400

        s3.upload_fileobj(file, bucket_name, file.filename)
        return jsonify({"message": f"File '{file.filename}' uploaded successfully to '{bucket_name}'"})

        except botocore.exceptions.NoCredentialsError:
        return jsonify({"error": "AWS credentials not found"}), 401

        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500


@app.route("/delete", methods=["POST"])
def delete_file():
        """Delete a file from a specified S3 bucket."""
        try:
        bucket_name = request.form.get("bucket_name")
        file_name = request.form.get("file_name")

        if not bucket_name or not file_name:
        return jsonify({"error": "Bucket name and file name are required"}), 400

        response = s3.list_objects_v2(Bucket=bucket_name, Prefix=file_name)
        if "Contents" not in response:
        return jsonify({"error": "File not found"}), 404

        s3.delete_object(Bucket=bucket_name, Key=file_name)
        return jsonify({"message": f"File '{file_name}' deleted successfully from '{bucket_name}'"})

        except botocore.exceptions.NoCredentialsError:
        return jsonify({"error": "AWS credentials not found"}), 401

        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500
```

```
@app.route("/copy_file", methods=["POST"])
def copy_file():
        """Copy a file within S3 from one bucket to another."""
        try:
        source_bucket = request.form.get("source_bucket")
        source_key = request.form.get("source_key")
        destination_bucket = request.form.get("destination_bucket")
        destination_key = request.form.get("destination_key")

        if not source_bucket or not source_key or not destination_bucket or not destination_key:
        return jsonify({"error": "All fields (source and destination) are required"}), 400

        copy_source = {"Bucket": source_bucket, "Key": source_key}

        # Copy the file to the destination bucket and key
        s3.copy_object(
        CopySource=copy_source,
        Bucket=destination_bucket,
        Key=destination_key
        )

        return jsonify({"message": f"File '{source_key}' successfully copied from '{source_bucket}' to
'{destination_bucket}/{destination_key}'"})

        except botocore.exceptions.NoCredentialsError:
        return jsonify({"error": "AWS credentials not found"}), 401

        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500


@app.route("/create_bucket", methods=["POST"])
def create_bucket():
        """Create a new S3 bucket."""
        try:
        bucket_name = request.form.get("bucket_name")
        region = request.form.get("region", "eu-north-1")

        if not bucket_name:
        return jsonify({"error": "Bucket name is required"}), 400
        if region == "us-east-1":
        s3.create_bucket(Bucket=bucket_name)
        else:
        s3.create_bucket(
        Bucket=bucket_name,
        CreateBucketConfiguration={'LocationConstraint': region}
        )
        return jsonify({"message": f"Bucket '{bucket_name}' created successfully"})
        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500



@app.route("/delete_bucket", methods=["POST"])
def delete_bucket():
        """Delete an S3 bucket."""
        try:
        bucket_name = request.form.get("bucket_name")

        if not bucket_name:
        return jsonify({"error": "Bucket name is required"}), 400

        s3.delete_bucket(Bucket=bucket_name)
        return jsonify({"message": f"Bucket '{bucket_name}' deleted successfully"})

        except botocore.exceptions.BotoCoreError as e:
        return jsonify({"error": f"AWS Error: {str(e)}"}), 500

        except Exception as e:
        return jsonify({"error": f"Something went wrong: {str(e)}"}), 500


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

User Interface:

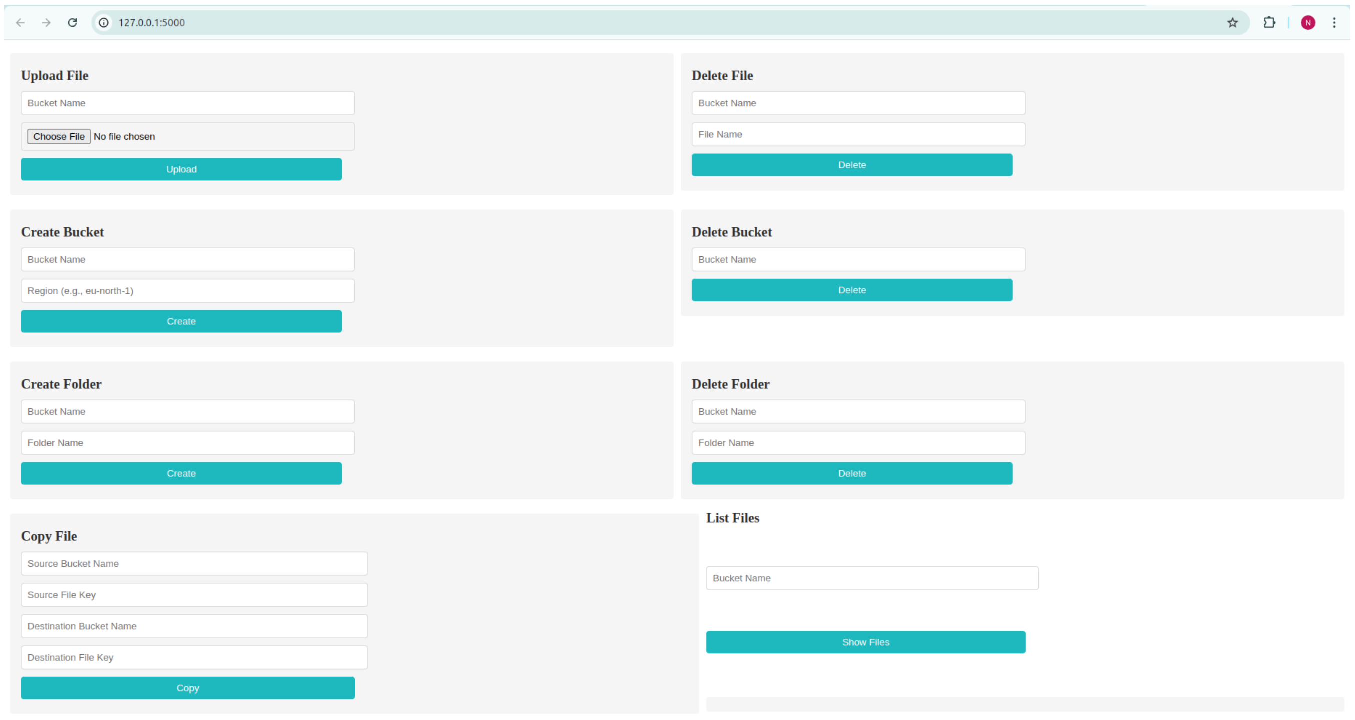
File : **index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>S3 File Manager</title>
        <style>
        .form-group {
        display: flex;
        justify-content: space-between;
        gap: 10px;
}

form {
        margin: 10px 0;
        width: 50%;
        height: 50%;
        display: flex;
        flex-direction: column;
        justify-content: space-between;
        background-color: #f8f8f8;
        padding: 15px;
        border-radius: 4px;
}

input, button {
        width: 50%;
        padding: 8px;
        margin: 5px 0;
        border: 1px solid #ddd;
        border-radius: 4px;
}

button {
        background-color: #22b9be;
        color: white;
        border: none;
}

pre {
        background-color: #f8f8f8;
        padding: 10px;
        border-radius: 4px;
        margin-top: 10px;
}

h3 {
        margin: 0;
        padding: 5px 0;
        text-align: left;
        color: #333;
}
        </style>
</head>
<body>
        <!-- Upload and Delete Files (Side by Side) -->
<div class="form-group">
        <form action="/upload" method="post" enctype="multipart/form-data">
        <h3>Upload File</h3>
        <input type="text" name="bucket_name" placeholder="Bucket Name" required>
        <input type="file" name="file" required>
        <button type="submit">Upload</button>
        </form>

        <form action="/delete" method="post">
        <h3>Delete File</h3>
        <input type="text" name="bucket_name" placeholder="Bucket Name" required>
        <input type="text" name="file_name" placeholder="File Name" required>
        <button type="submit">Delete</button>
        </form>
</div>

<!-- Create and Delete Bucket (Side by Side) -->
<div class="form-group">
        <form action="/create_bucket" method="post">
        <h3>Create Bucket</h3>
        <input type="text" name="bucket_name" placeholder="Bucket Name">
        <input type="text" name="region" placeholder="Region (e.g., eu-north-1)">
        <button type="submit">Create</button>
        </form>
```

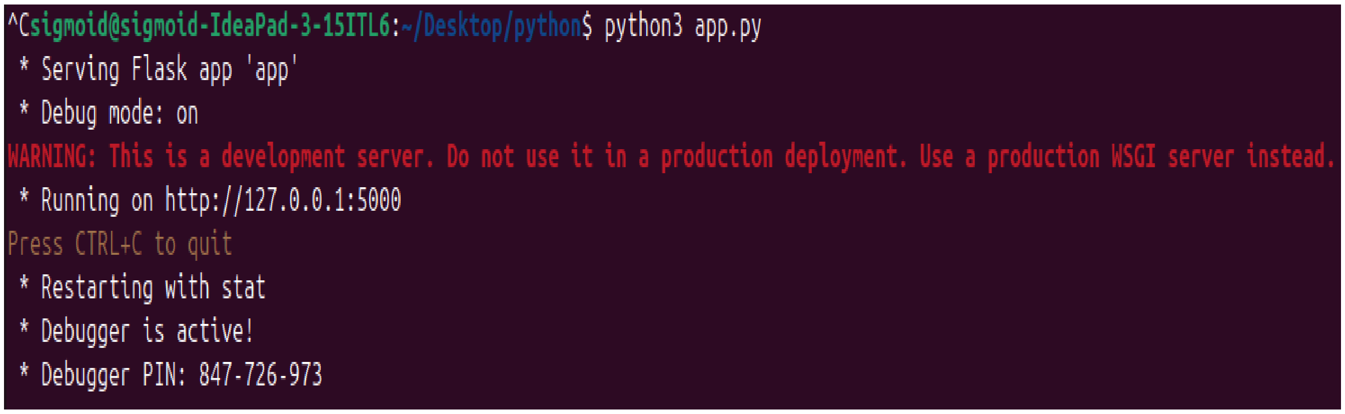
```
            <form action="/delete_bucket" method="post">
            <h3>Delete Bucket</h3>
            <input type="text" name="bucket_name" placeholder="Bucket Name" required>
            <button type="submit">Delete</button>
            </form>
</div>

<!-- Create and Delete Folder (Side by Side) -->
<div class="form-group">
            <form action="/create_folder" method="post">
            <h3>Create Folder</h3>
            <input type="text" name="bucket_name" placeholder="Bucket Name" required>
            <input type="text" name="folder_name" placeholder="Folder Name" required>
            <button type="submit">Create</button>
            </form>

            <form action="/delete_folder" method="post">
            <h3>Delete Folder</h3>
            <input type="text" name="bucket_name" placeholder="Bucket Name" required>
            <input type="text" name="folder_name" placeholder="Folder Name" required>
            <button type="submit">Delete</button>
            </form>
</div>

<!-- Copy File and List Files (Side by Side) -->
<div class="form-group">
            <form action="/copy_file" method="post">
            <h3>Copy File</h3>
            <input type="text" name="source_bucket" placeholder="Source Bucket Name" required>
            <input type="text" name="source_key" placeholder="Source File Key" required>
            <input type="text" name="destination_bucket" placeholder="Destination Bucket Name"
required>
            <input type="text" name="destination_key" placeholder="Destination File Key" required>
            <button type="submit">Copy</button>
            </form>

            <div style="flex: 1; display: flex; flex-direction: column; justify-content: space-between;">
            <h3>List Files</h3>
            <input type="text" id="bucket_name" placeholder="Bucket Name" required>
            <button onclick="listFiles()">Show Files</button>
            <pre id="file-list"></pre>
            </div>
</div>
```

```
        <script>
        async function listFiles() {
        let bucketName = document.getElementById("bucket_name").value;
        if (!bucketName) {
        alert("Please enter a bucket name");
        return;
        }
        let response = await fetch(`/list?bucket_name=${bucketName}`);
        let files = await response.json();
        document.getElementById("file-list").innerText = JSON.stringify(files, null, 2);
        }
        </script>
</body>
</html>
```

User Interface:

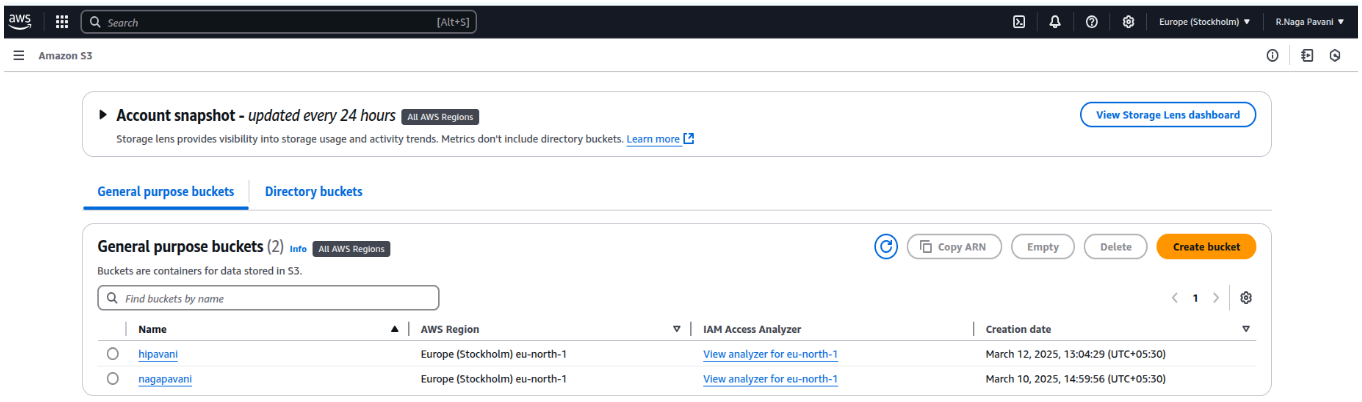


Running Flask S3 File Manager Application on Localhost:



```
^Csigmoid@sigmoid-IdeaPad-3-15ITL6:~/Desktop/python$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 847-726-973
```

Buckets stored in s3:



Account snapshot - *updated every 24 hours* All AWS Regions

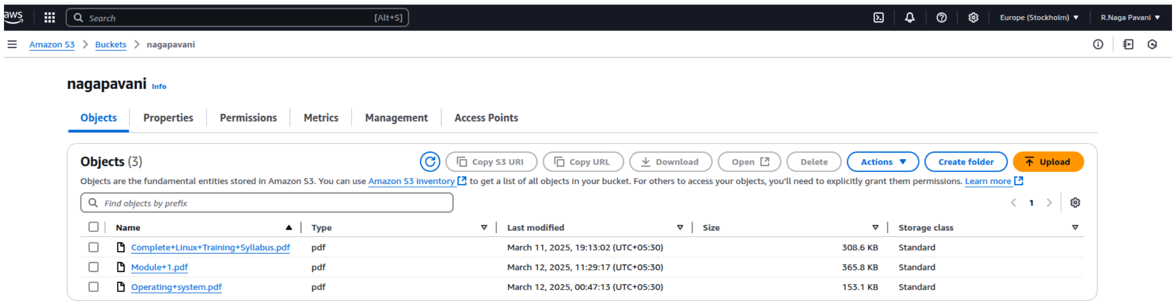
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more

General purpose buckets (2) Info All AWS Regions

Buckets are containers for data stored in S3.

	Name	AWS Region	IAM Access Analyzer	Creation date
○	hipavani	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	March 12, 2025, 13:04:29 (UTC+05:30)
○	nagapavani	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	March 10, 2025, 14:59:56 (UTC+05:30)

Files stored in the 'nagapavani' S3 Bucket:



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
Complete+Linux+Training+Syllabus.pdf	pdf	March 11, 2025, 19:13:02 (UTC+05:30)	308.6 KB	Standard
Module+1.pdf	pdf	March 12, 2025, 11:29:17 (UTC+05:30)	365.8 KB	Standard
Operating+system.pdf	pdf	March 12, 2025, 00:47:13 (UTC+05:30)	153.1 KB	Standard