Vijaya Sekhar Yeleswarapu - Assignment

Wednesday, October 2, 2024 2:58 PM

Task1 :

- Think of API that could be part of this application
- Build a test cases to test API
- Mock the data for inputs and expected outputs
- Write automation code to test these API

Even one API tested through automation is good enough. Please also know that there is no right or wrong answer here. I am loo king for the candidate creativity and depth

Based on understanding of the task1.

1. A group can only be associated with one folder, but users can belong to many groups., groups are associated with Folders
2. Each group has one permission set (read, editor, or admin), which defines what actions users can perform on the associated fo lder.

 # High Level Test Design

Group Association with Folder :

- Verify API associates a group with a folder.
- Verify that a group can only be associated with **one** folder.
- Verify that associating a group with another folder returns an error or prevents the action.

User Association with Group :

- Verify that API allows a user to be part of multiple groups.
- Verify that users get permissions based on the group they belong to for the associated folder.

User Read Permissions:

- Verify users can view and access the folder.
- Verify users can only see the files in the folder (no editing or adding).

- **User Editor Permissions**:
  - Verify users can view, access, add, and delete files in the folder.
  - Verify users cannot create subfolders or delete the folder itself.

- **User Admin Permissions**:
  - Verify users can **create subfolders**, **delete folders**, **add and delete files**.

User Access Control :

- Verify that a user's actions are limited to their group's permission set.
- Verify access based on the group a user belongs to and the associated folder.
- Verify a user in a "read" group cannot delete files, while a user in an "admin" group can delete folders.

 # Model Test cases for API

 # Smoke Test Cases

These test cases will cover the end to end functionality of the API.

1) Verify Group Association

- **API**: POST /group/associate-folder
- **Description**: Ensure that a group is associated with only one folder.
- **Precondition**: Group A and folders X and Y are already created.
- **Test Data**:
  - Group: A
  - Folder: X, Y

Test Step	Action	Expected Result
1	Associate Group A with Folder X using API POST /group/associate-folder.	Group A is successfully associated with Folder X.
2	Attempt to associate Group A with Folder Y using API POST /group/associate-folder.	The association fails, or an error message is returned.
3	Validate the response from step 2.	The response confirms that Group A cannot be associated with multiple folders.

2) Verify User Permissions - Read Only

- **API**: GET /folder/{folderId}/files
- **Description**: Ensure users with "read" permissions can only view files in the folder.

- **Precondition**: Folder A exists, and User X is added to Group A with "Read" permissions.
- **Test Data**:
  - User: X
  - Group: A
  - Folder: A

Test Step	Action	Expected Result
1	Add User X to Group A with "Read" permissions.	User X is successfully added to Group A.
2	Ensure User X can view and access Folder A using API GET /folder/{folderId}/files.	The list of files in Folder A is successfully retrieved.
3	Try to add a file to Folder A as User X.	The file addition fails, or an error message is returned.
4	Validate the response from step 3.	The response confirms that User X has "Read Only" permissions and cannot add files.

3) Verify User Permissions - Edit

- **API**: POST /folder/{folderId}/file
- **Description**: Ensure users with "editor" permissions can add and delete files but cannot delete folders.
- **Precondition**: Folder B exists, and User Y is added to Group B with "Edit" permissions.
- **Test Data**:
  - User: Y
  - Group: B
  - Folder: B

Test Step	Action	Expected Result
1	Add User Y to Group B with "Editor" permissions.	User Y is successfully added to Group B.
2	Ensure User Y can add a file to Folder B using API POST /folder/{folderId}/file.	The file is successfully added to Folder B.
3	Ensure User Y can delete a file from Folder B using API DELETE /folder/{folderId}/file/{fileId}.	The file is successfully deleted from Folder B.
4	Try to delete Folder B as User Y using API DELETE /folder/{folderId}.	The deletion fails, or an error message is returned.
5	Validate the response from step 4.	The response confirms that User Y cannot delete the folder.

4) Verify User Permissions - Admin

- **API**: DELETE /folder/{folderId}
- **Description**: Ensure users with "admin" permissions can create subfolders, delete folders, and manage files.
- **Precondition**: Folder C exists, and User Z is added to Group C with "Admin" permissions.
- **Test Data**:
  - User: Z
  - Group: C
  - Folder: C

Test Step	Action	Expected Result
1	Add User Z to Group C with "Admin" permissions.	User Z is successfully added to Group C.
2	Ensure User Z can create a subfolder in Folder C using API POST /folder/{folderId}/subfolder.	A subfolder is successfully created within Folder C.
3	Ensure User Z can delete Folder C using API DELETE /folder/{folderId}.	Folder C is successfully deleted.
4	Ensure User Z can add and delete files in Folder C using APIs POST /folder/{folderId}/file and DELETE /folder/{folderId}/file/{fileId}.	Files are successfully added and deleted in Folder C.

Regression Test Cases

These test cases cover some of the contract testing, response time validation, security testing, positive and negative valida tions.

#1. Contract Testing

1) Verify Response Schema for User Accessing a Folder

- **Test Case ID**: TC001
- **API Endpoint**: GET /user/{userId}/folders
- **Test Objective**: To validate that the response schema for fetching accessible folders by a user matches the expected format.

Pre-Conditions:

a. User with a valid userId exists.
b. Folders are assigned to the user.

Test Data:

- userId: <valid_user_id>

Expected Response Schema:

```
{
 "id": "number",
 "name": "string",
 "groupId": "number"
}
```

Steps:

a. Send a GET request to fetch folders accessible by a user using the endpoint /user/{userId}/folders.
b. Validate that the response matches the expected schema.

Assertions:

- Validate that the response contains all required fields (e.g., id, name, groupId).
- Validate the data types of each field (id should be a number, name should be a string, groupId should be a number).

Expected Result:

The response should contain all required fields and the data types should match the expected schema.

2) Validate the Response Schema for Group Permissions

- **Test Case ID**: TC002
- **API Endpoint**: GET /groups/{groupId}/permissions
- **Test Objective**: To validate the response schema for fetching group permissions and ensuring permissionSet values are correct.

Pre-Conditions:

a. Group with a valid groupId exists.
b. The group has been assigned permissions.

Test Data:

- groupId: <valid_group_id>

Expected Response Schema:

```
{
 "groupId": "number",
 "permissionSet": "string",
 "folderId": "number"
}
```

Steps:

a. Send a GET request to fetch the permissions of a group using the endpoint /groups/{groupId}/permissions.
b. Ensure that the permissionSet contains one of the valid values: read, editor, or admin.

Assertions:

- Validate that the permissionSet field contains only valid permission types (read, editor, admin).

- Validate that folderId is correctly associated with the group.

Expected Result:
The response should contain valid permission types, and the folderId should be correctly associated with the group.

#Response Time Validation

3) **Validate Response Time for Folder Access API**
   - **Test Case ID**: TC003
   - **API Endpoint**: GET /user/{userId}/folders
   - **Test Objective**: To validate that the response time for accessing folders is within an acceptable threshold.

Pre-Conditions:
a. A valid userId exists, and folders are assigned to the user.

Test Data:
- userId: <valid_user_id>

Steps:
a. Send a GET request to /user/{userId}/folders.
b. Measure the response time.
c. Set a threshold of 500ms.

Assertions:
- The API response time should be less than or equal to 500ms.
- The test should log the response time and assert it is below the threshold.
- Separate thresholds may be defined for different environments (e.g., lower threshold for production).

Expected Result:
The API should respond within the defined threshold (500ms), with no significant delay.

Validate Response Time for File Deletion API
- **Test Case ID**: TC004
- **API Endpoint**: DELETE /folder/{folderId}/files/{fileId}
- **Test Objective**: To validate that the response time for deleting a file is within an acceptable threshold.

Pre-Conditions:
a. A valid folderId and fileId exist, and the user has permissions to delete the file.

Test Data:
- folderId: <valid_folder_id>
- fileId: <valid_file_id>

Steps:
a. Send a DELETE request to /folder/{folderId}/files/{fileId}.
b. Measure the response time.
c. Set a threshold of 300ms.

Assertions:
- The API should complete the delete action within the set threshold (300ms).
- If the API times out, it should return a 504 Gateway Timeout error.

Expected Result:
The file should be deleted successfully within the 300ms threshold, or a 504 error should be returned if the threshold is exceeded.

2. Security Testing

4) **# Validate Unauthorized Access to Protected Folder**
   - **Test Case ID**: TC005
   - **API Endpoint**: GET /folder/{folderId}/files
   - **Test Objective**: To ensure that unauthorized users cannot access protected folders.

Pre-Conditions:
a. The folder is protected, and the user does not belong to the group with access.

Test Data:
- folderId: <protected_folder_id>
- Unauthorized userId: <invalid_user_id>

Steps:
a. Attempt to access the folder using the unauthorized userId by sending a GET request to /folder/{folderId}/files.

Assertions:
- The API should return 403 Forbidden for unauthorized access attempts.
- No sensitive data (e.g., folder contents) should be leaked in the response.

Expected Result:
The API should deny access with a 403 status code, and no sensitive data should be exposed.

Validate Injection Attack Protection
- **Test Case ID**: TC006
- **API Endpoint**: POST /folder/{folderId}/files
- **Test Objective**: To validate that the API protects against SQL injection attacks.

Pre-Conditions:
a. A valid folder exists with write permissions for testing.

Test Data:
- folderId: <valid_folder_id>
- Malicious payload: 'DROP TABLE files;'

Steps:
a. Send a POST request to /folder/{folderId}/files with the file name field containing a SQL injection payload (e.g., 'DROP TABLE files;').

Assertions:
- The API should sanitize the input and prevent SQL injection attacks.
- The API should return a 400 Bad Request for invalid inputs.

Expected Result:
The API should reject the malicious input and respond with a 400 Bad Request.

5) **# Validate Sensitive Data Exposure**
   - **Test Case ID**: TC007
   - **API Endpoint**: GET /user/{userId}
   - **Test Objective**: To ensure that sensitive information is not exposed in the API response.

Pre-Conditions:
a. The user's sensitive data (e.g., passwords, tokens, PII) is available in the database but should not be returned by the API.

Test Data:
- userId: <valid_user_id>

Steps:
a. Send a GET request to /user/{userId}.
b. Validate that sensitive information like passwords, tokens, or PII is not included in the response.

Assertions:
- Sensitive information should not be exposed in the API response.
- Only necessary fields should be returned.

Expected Result:
No sensitive data should be returned, and the API should only provide essential user details.

 # Positive and Negative Test Cases

 # Positive Add a New File with Valid Data
- **Test Case ID**: TC008
- **API Endpoint**: POST /folder/{folderId}/files
- **Test Objective**: To validate that a new file can be successfully added with valid data.

Pre-Conditions:
a. The user has valid permissions to add files to the folder.

Test Data:

{
"fileName": "newfile.txt",
"userId": 1
}

Steps:

a. Send a POST request to /folder/{folderId}/files with the above request body.
b. Validate that the file is successfully added.

Assertions:

- The API should return HTTP 201 Created.
- Validate that the file is present in the folder.

Expected Result:

The file should be successfully added to the folder.

 # Positive Delete a File with Valid Credentials

- **Test Case ID**: TC009
- **API Endpoint**: DELETE /folder/{folderId}/files/{fileId}
- **Test Objective**: To validate that a file can be successfully deleted with valid credentials.

Pre-Conditions:

a. The user has valid permissions to delete the file.

Test Data:

- folderId: <valid_folder_id>
- fileId: <valid_file_id>

Steps:

a. Send a DELETE request to /folder/{folderId}/files/{fileId}.
b. Validate that the file is deleted successfully.

Assertions:

- The API should return HTTP 200 OK.
- Validate that the file no longer exists in the folder.

Expected Result:

The file should be successfully deleted.

 # Negative : Add a File to a Folder Without Permissions

- **Test Case ID**: TC0010
- **API Endpoint**: POST /folder/{folderId}/files
- **Test Objective**: To validate that adding a file without proper permissions fails.

Pre-Conditions:

a. The user does not have write permissions for the folder.

Test Data:

{
"fileName": "unauthorizedfile.txt",
"userId": 1
}

Steps:

a. Send a POST request to /folder/{folderId}/files without proper permissions.

Assertions:

- The API should return 403 Forbidden.
- The file should not be added to the folder.

Expected Result:

The API should deny the action with a 403 status code.

 # Negative : Delete a Folder Without Admin Rights

- **Test Case ID**: TC0011
- **API Endpoint**: DELETE /folder/{folderId}
- **Test Objective**: To validate that deleting a folder without admin rights fails.

Pre-Conditions:

a. The user has read or editor permissions but not admin rights.

Test Data:

- folderId: <valid_folder_id>

Steps:

a. Send a DELETE request to /folder/{folderId} as a non-admin user.

Assertions:

- The API should return 403 Forbidden.
- The folder should not be deleted.

Expected Result:

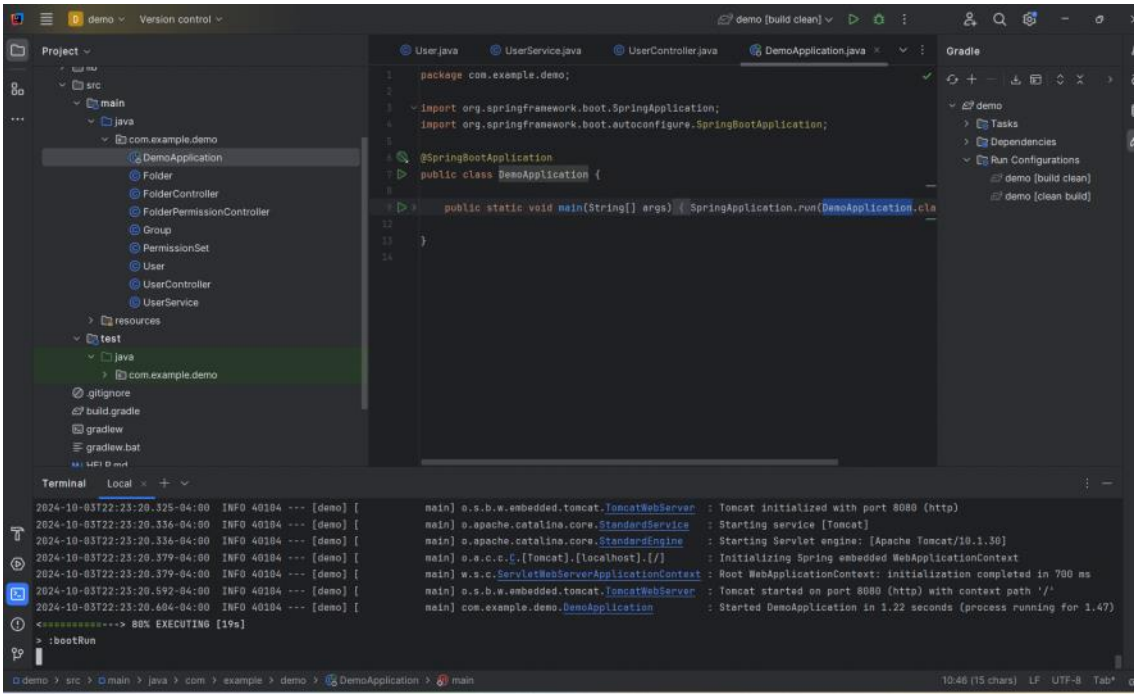
The folder should not be deleted, and the API should return 403

 # Mocking Framework

Created a Gradle spring boot Framework Project for mocking input / output and to test API.

Created Model classes for Group, Folder, User, PermissionSet.
Created Controller classes for mocking responses.

Run command - ./gradlew clean build , to build the application
Run Command - ./gradlew bootRun , to up the Mock application services.



 # API Automation Framework

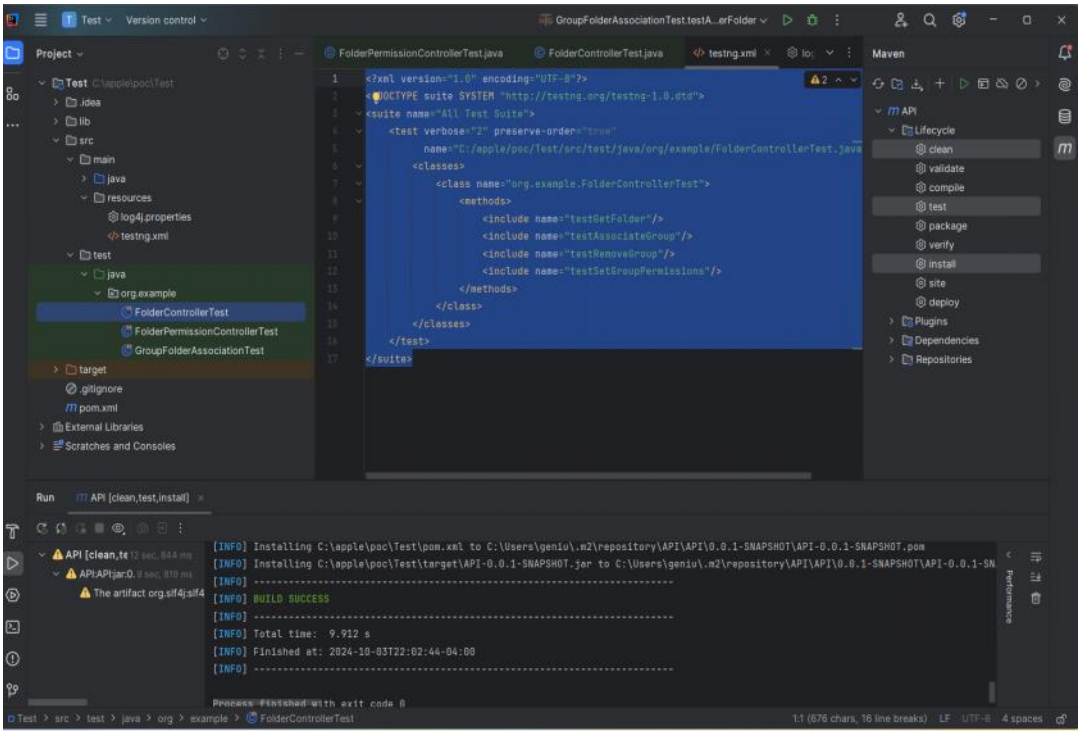
This Project is built for testing Application API's using mocked services.

Used Java 18 , TestNG, RestAssured , Maven to test the Application API.

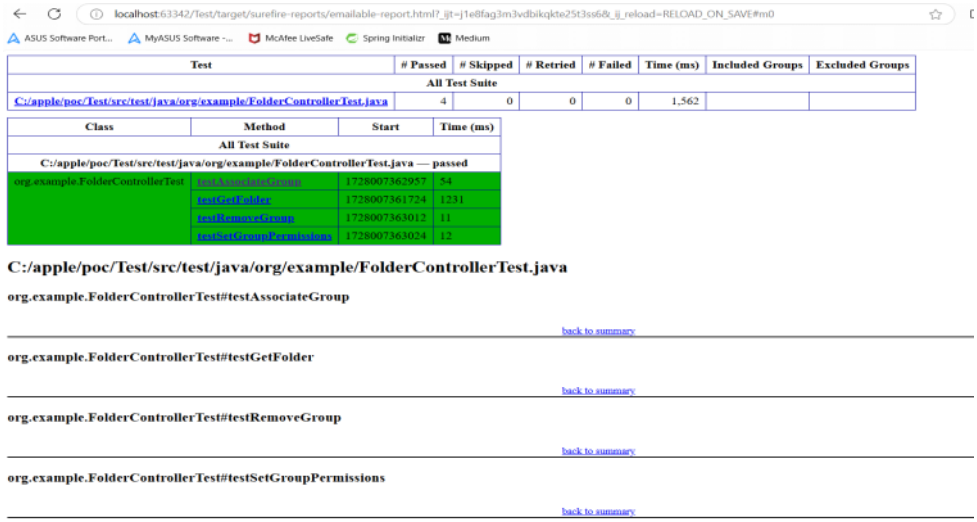
Execution is Done Through TestNG.xml file.

There are Four Tests Classes developed as below to test the associated test cases for Folders, Permission, GroupFolderAssociation.

Below is testNG Test class for testing Folder related API scenarios, like GetFolder, AssociateGroup, RemoveGroup,GroupPermissions.



 # Test Results :



Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
All Test Suite							
C:/apple/poc/Test/src/test/java/org/example/FolderControllerTest.java	4	0	0	0	1,562		

Class	Method	Start	Time (ms)
All Test Suite			
C:/apple/poc/Test/src/test/java/org/example/FolderControllerTest.java — passed			
org.example.FolderControllerTest	testAssociateGroup	1728007362957	54
	testGetFolder	1728007361724	1231
	testRemoveGroup	1728007363012	11
	testSetGroupPermissions	1728007363024	12

C:/apple/poc/Test/src/test/java/org/example/FolderControllerTest.java

org.example.FolderControllerTest#testAssociateGroup

back to summary

org.example.FolderControllerTest#testGetFolder

back to summary

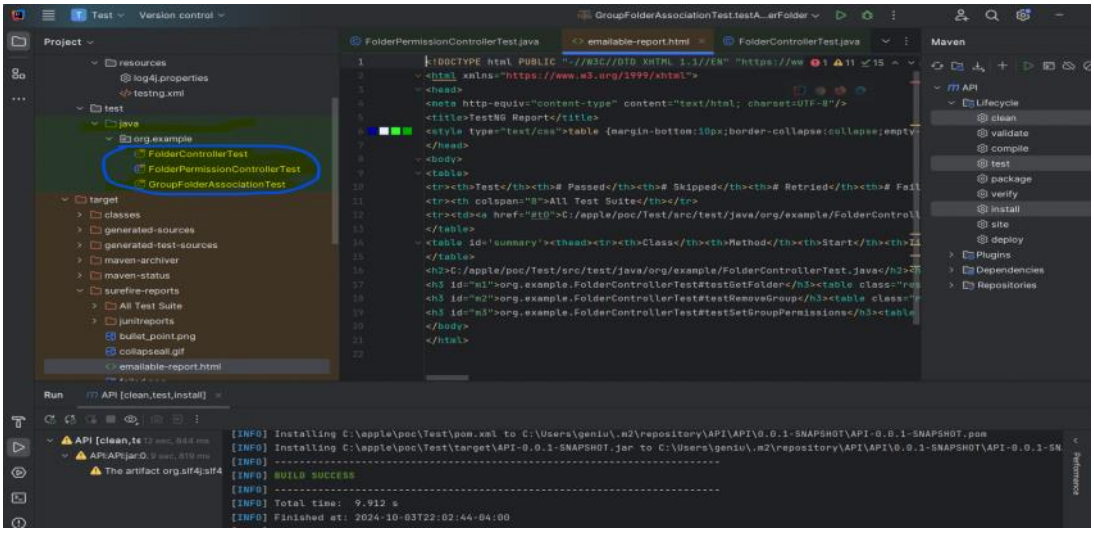
org.example.FolderControllerTest#testRemoveGroup

back to summary

org.example.FolderControllerTest#testSetGroupPermissions

back to summary

Conclusion : Will provide the demo for the highlighted API Automated test cases.



Task 2: Use the input.csv

Write a simple program in a language of Arslan Choice to generate the following output

403|1
200|2
500|1
401|1

Please note there may be other status number in Column C in real data.

Solution : To test the source input.csv file , written pyspark in Glue code and ran the Glue Job pipeline with parameters. It will take the input file from designated S3 bucket where input.csv file exists and place results into the folder in the designated S3 bucket, In new folder "valid_status_counts", as shown in the below diagram.

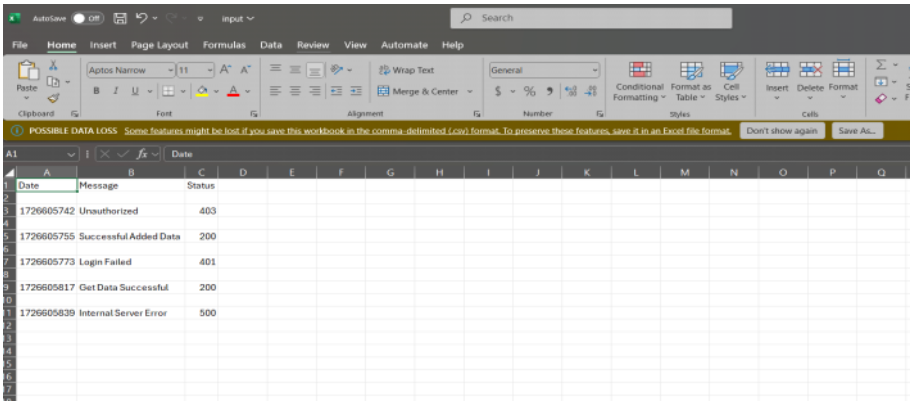
Below is the pyspark program, used to test the imput.csv file.

```
import sys
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
import re


sc = SparkContext()
glueContext = GlueContext(sc)
```

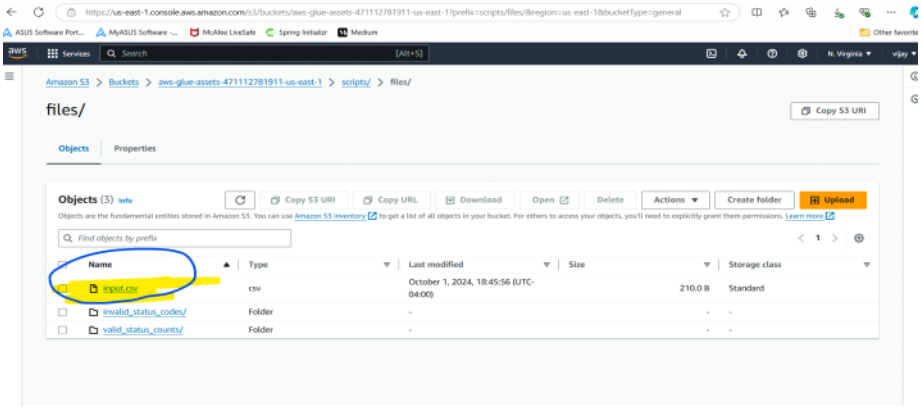
```
spark = glueContext.spark_session


args = getResolvedOptions(sys.argv, ['input_path', 'output_path'])

input_path = args['input_path']
output_path = args['output_path']

print("input_path :"+input_path)
print("output_path :"+output_path)

VALID_STATUS_CODE_REGEX = r"^(100|[1-5][0-9]{2})$"

df = spark.read.csv(input_path, header=True)


valid_df = df.filter(col('status_code').rlike(VALID_STATUS_CODE_REGEX))
invalid_df = df.filter(~col('status_code').rlike(VALID_STATUS_CODE_REGEX))

valid_status_count_df = valid_df.groupBy('status_code').agg(count('status_code').alias('count'))

valid_status_count_df.write.csv(f"{output_path}/valid_status_counts", header=True)

invalid_df.write.csv(f"{output_path}/invalid_status_codes", header=True)

print("Valid Status Codes Count:")
valid_status_count_df.show()

print("Invalid Status Codes:")
invalid_df.show()


sc.stop()
```

TestResult

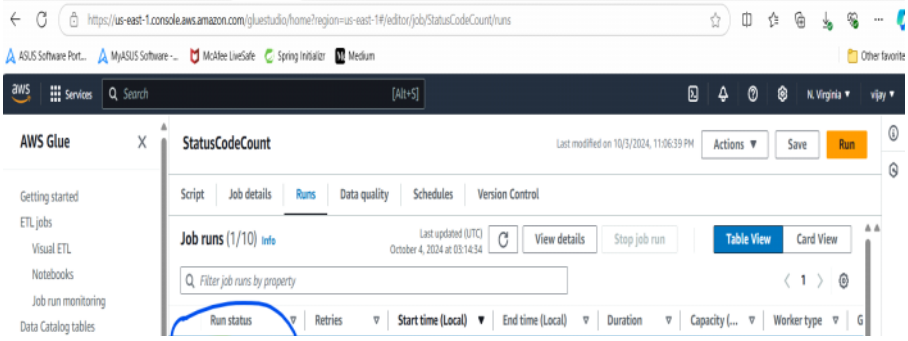
Input file Given in S3:

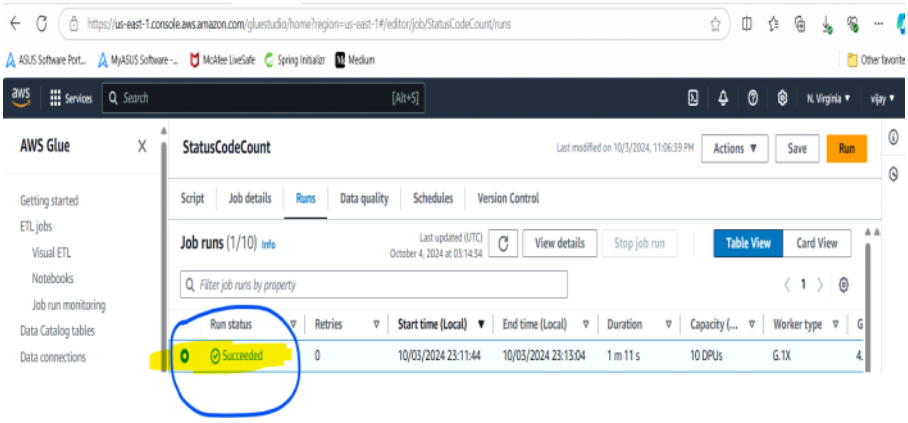


Date	Message	Status
1726605742	Unauthorized	403
1726605755	Successful Added Data	200
1726605773	Login Failed	401
1726605817	Get Data Successful	200
1726605839	Internal Server Error	500



Success Run Status





Valid Status Codes Output :