Coding Interview [GA4GH Developer]

Reporting manual testing (run_tests.py)

Introduction:
The code provided is a Python script that performs testing for compliance with the Data Repository Service (DRS) v1.2 specification. The DRS is a standard API for accessing and retrieving data objects in a distributed environment. The script makes use of the 'requests' library for making HTTP requests to the DRS endpoint, 'jsonschema' library for validating JSON responses, and 'logging' library for logging test results.

Testing Approach:
The script follows a functional testing approach, where it sends HTTP requests to the DRS endpoint and validates the response against expected criteria defined by the DRS v1.2 specification. The script includes the following tests:

Test for Successful Request: This test validates if a successful response is received when retrieving a DRS object with a valid object ID. It checks if the HTTP status code is 200, indicating a successful request.

Test for Content Type: This test validates if the 'Content-Type' header in the response is set to 'application/json', as specified by the DRS v1.2 specification.

Test for Successful Schema: This test validates if the 'self_uri' field in the response JSON contains the same object ID as requested, and if the HTTP status code is 200. This ensures that the returned DRS object ID matches the requested ID and the response is successful.

Test for Access Methods: This test validates if the response JSON contains all the expected fields required by the DRS v1.2 specification, including 'access_methods', 'checksums', 'created_time', 'description', 'id', 'mime_type', 'name', 'size', 'updated_time', and 'version'.

Each test is performed using the 'call_api()' function, which sends an HTTP request to the DRS endpoint with a specific object ID and captures the response. The response is then validated against the expected criteria using the 'print_response()' function, which logs the test result with the object ID, test name, pass/fail status, and a relevant message.

Test Results:
The script uses the 'run_tests()' function to run the tests for a list of object IDs defined in the 'SUCCESS_STATUS_OBJECTS' variable. The 'passed_tests' variable keeps track of the number of tests that pass for each object ID, and the total number of passed tests is logged at the end using the 'logging' library.

Conclusion:
The script provides a functional testing approach to validate compliance with the DRS v1.2 specification for retrieving DRS objects. It checks for successful responses, correct content type, matching object ID, and presence of expected fields in the response JSON. The logging of test results provides a summary of the number of tests passed and failed, indicating if all tests have passed or if there are some failures. This script can be used as a foundation for further testing and validation of DRS implementations to ensure compliance with the DRS v1.2 specification.

Reporting unit testing (test_endpoints.py)

This is a Python script that contains a unit test class TestEndpoints with several test methods to test endpoints of a DRS (Data Repository Service) API. Here's a summary of what each test method does:

test_successful_request: Sends GET requests to the endpoint URL with different object IDs from SUCCESS_STATUS_OBJECTS list and checks if the response status code is 200, indicating a successful request.

test_failed_request: Sends GET requests to the endpoint URL with different object IDs from FAILURE_STATUS_OBJECTS list and checks if the response status code is 404, indicating a failed request for nonexistent objects.

test_content_type: Sends GET requests to the endpoint URL with different object IDs from DRS_OBJECTS list and checks if the 'Content-Type' header in the response is 'application/json', indicating that the response is in JSON format.

test_success_schema: Sends GET requests to the endpoint URL with different object IDs from SUCCESS_STATUS_OBJECTS list, retrieves the response as JSON, and validates it against a JSON schema (DRS_SCHEMA) using jsonschema.validate() method. If validation fails, the test fails with an error message.

test_failure_schema: Sends GET requests to the endpoint URL with different object IDs from FAILURE_STATUS_OBJECTS list, retrieves the response as JSON, and expects schema validation to fail for nonexistent objects, so no action is needed.

test_missing_access_url_or_id: Sends GET requests to the endpoint URL with different object IDs from SUCCESS_STATUS_OBJECTS list, retrieves the response as JSON, and checks if the 'access_url' and 'access_id' fields are not present in the response, indicating that the access API did not return unexpected fields.

test_access_id_with_valid_access_url: Sends GET requests to the endpoint URL with different object IDs from SUCCESS_STATUS_OBJECTS list, retrieves the response as JSON, gets the 'access_url' from the response, and sends a GET request to the access URL to check if it returns a 200 status code, indicating a valid access URL.

test_access_id_with_invalid_access_url: Sends GET requests to the endpoint URL with different object IDs from SUCCESS_STATUS_OBJECTS list, retrieves the response as JSON, gets the 'access_id' from the response, and sends a GET request to an invalid access URL to check if it does not return a 200 status code, indicating an invalid access URL.

Note: The script assumes that there are some variables/constants such as endpoint_url, SUCCESS_STATUS_OBJECTS, FAILURE_STATUS_OBJECTS, DRS_OBJECTS, and DRS_SCHEMA defined or imported from other modules, and utils.py module is imported for utility functions. Please make sure to provide the necessary dependencies and variables for the script to run successfully.