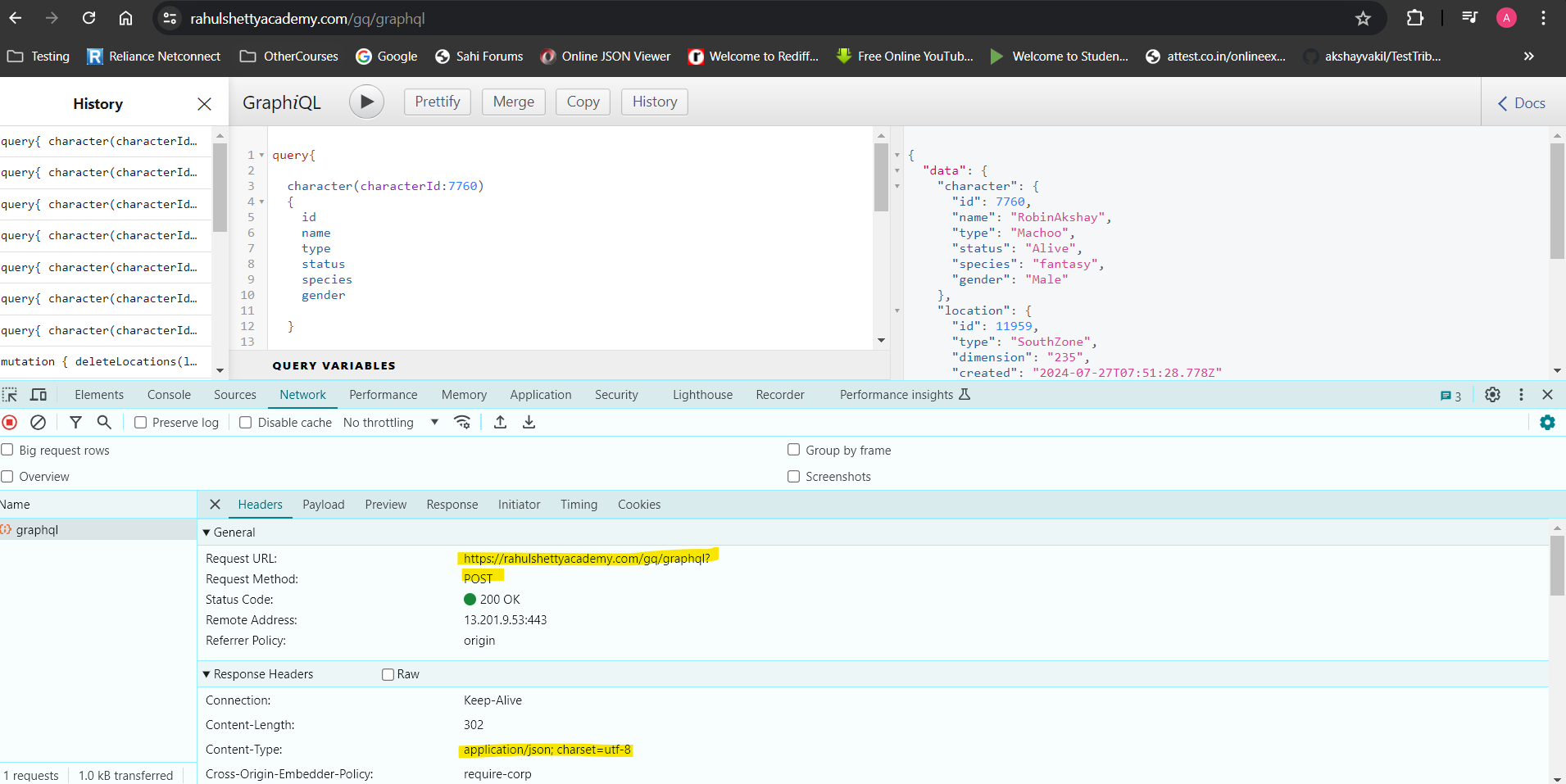
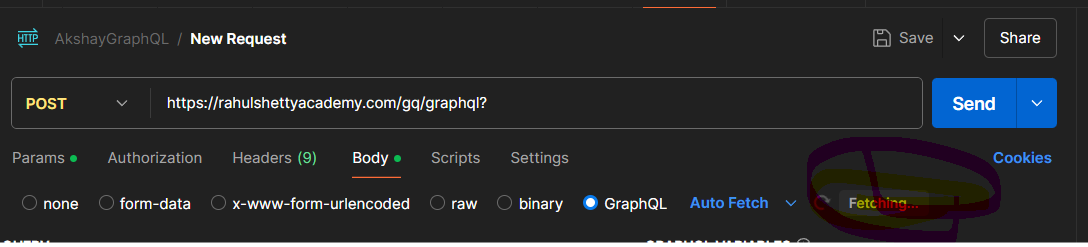
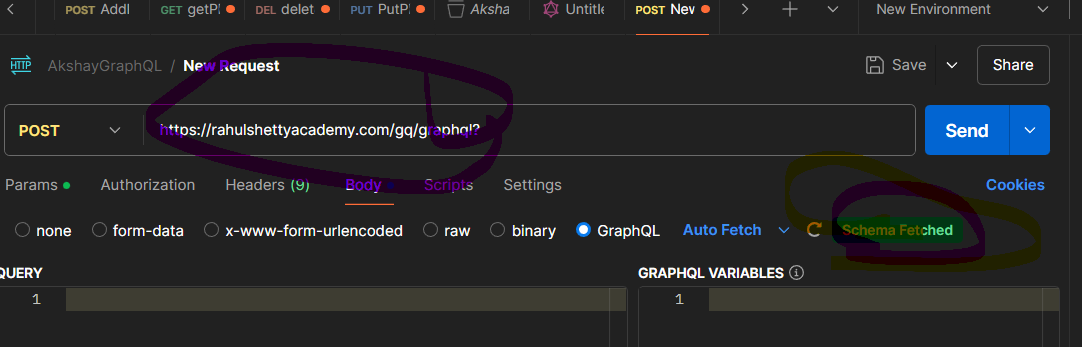
* **ByDefault all GRaphQL calls are POST calls**
* **End Point can be obtained from browser console**

****

* **Scehma will be fetched automatically when we put end endpot in POSTMAN tool**

****

****

**1. What is GraphQL and how does it differ from REST?**

**Explanation:** GraphQL allows clients to request exactly the data they need via a single endpoint, unlike REST which uses multiple endpoints for different resources. This reduces over-fetching and under-fetching of data.

**Rest Assured Code:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testFetchUserData() {

String query = "{ \"query\": \"{ user(id: \\\"1\\\") { name email } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertEquals(response.jsonPath().getString("data.user.name"), "John Doe");

}

}

**2. How do you perform GraphQL API testing in an automated test suite?**

**Explanation:** Automated GraphQL testing involves sending queries or mutations to the GraphQL endpoint and validating the responses. This can be achieved using Rest Assured by constructing appropriate requests and asserting the response.

**Rest Assured Code:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testFetchUserData() {

String query = "{ \"query\": \"{ user(id: \\\"1\\\") { name email } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertEquals(response.jsonPath().getString("data.user.name"), "John Doe");

}

}

**3. What is the purpose of fragments in GraphQL, and how would you test them?**

**Explanation:** Fragments in GraphQL allow you to reuse parts of queries, reducing redundancy. Testing involves ensuring that fragments are included correctly and that their fields are returned as expected.

**Rest Assured Code:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testFetchUserDataWithFragment() {

String query = "{ \"query\": \"fragment UserFields on User { name email } query { user(id: \\\"1\\\") { ...UserFields } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertEquals(response.jsonPath().getString("data.user.name"), "John Doe");

}

}

**4. How do you handle pagination in GraphQL queries, and how would you test it?**

**Explanation:** Pagination in GraphQL is managed using arguments like first, after, last, and before. Testing ensures that these arguments correctly paginate data and handle edge cases like empty results.

**Rest Assured Code:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testFetchPaginatedUsers() {

String query = "{ \"query\": \"query { users(first: 5, after: \\\"cursor123\\\") { edges { node { id name } } pageInfo { hasNextPage } } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertTrue(response.jsonPath().getList("data.users.edges").size() <= 5);

Assert.assertTrue(response.jsonPath().getBoolean("data.users.pageInfo.hasNextPage"));

}

}

**5. What are mutations in GraphQL, and how do you validate their correctness in automated tests?**

**Explanation:** Mutations in GraphQL are used to modify server-side data, such as creating or updating records. Validation involves ensuring the mutation performs the expected changes and returns the correct results.

**Rest Assured Code:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testCreateUser() {

String query = "{ \"query\": \"mutation { createUser(input: { name: \\\"Jane Doe\\\", email: \\\"jane.doe@example.com\\\" }) { user { id name } } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertEquals(response.jsonPath().getString("data.createUser.user.name"), "Jane Doe");

}

}

**6. Can you explain how you would test for authorization and authentication in a GraphQL API?**

**Explanation:** Testing for authorization and authentication involves ensuring that only authenticated users can access protected resources and that permissions are enforced correctly. This includes handling both valid and invalid tokens.

**Rest Assured Code:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testUnauthorizedAccess() {

String query = "{ \"query\": \"query { privateData { sensitiveField } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 401); // Unauthorized

Assert.assertTrue(response.jsonPath().getString("errors[0].message").contains("Unauthorized"));

}

@Test

public void testAuthorizedAccess() {

String query = "{ \"query\": \"query { privateData { sensitiveField } }\" }";

Response response = RestAssured.given()

.contentType("application/json")

.header("Authorization", "Bearer valid-token")

.body(query)

.post("http://localhost:4000/graphql");

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertNotNull(response.jsonPath().getString("data.privateData.sensitiveField"));

}

}

**7. How do you test for query complexity and performance in a GraphQL API?**

**Explanation:** Testing query complexity and performance involves ensuring that queries are optimized and execute within acceptable time limits. Performance testing checks response times, while complexity tests ensure queries do not exceed defined limits.

**Rest Assured Code for Performance:**

java

Copy code

import io.restassured.RestAssured;

import io.restassured.response.Response;

import org.testng.Assert;

import org.testng.annotations.Test;

public class GraphQLTest {

@Test

public void testQueryPerformance() {

String query = "{ \"query\": \"query { complexField { deeplyNestedField { anotherNestedField { finalField } } } }\" }";

long startTime = System.currentTimeMillis();

Response response = RestAssured.given()

.contentType("application/json")

.body(query)

.post("http://localhost:4000/graphql");

long duration = System.currentTimeMillis() - startTime;

Assert.assertEquals(response.getStatusCode(), 200);

Assert.assertTrue(duration < 2000); // e.g., query should complete within 2 seconds

}

}

These explanations and code snippets provide a concise overview of how to approach each GraphQL testing scenario using Rest Assured.