Integration Test - SOAP

In this lesson, we will learn how to call a SOAP web service, chain multiple web service calls, automate and write an integration test.

We'll cover the following

Integration scenario

In this scenario, we are simulating and automating the integration flow of a single service.

We will use our demo SOAP web services to automate the below integration flow as follows:

1. Create a new `Student`
2. Verify `Student` is created
3. Search the newly-created `Student` by `id`
4. Update the created `Student` with new information
5. Verify the search result
6. Delete the created `Student`
7. Verify the `Student` is deleted

```
import static org.testng.Assert.assertEquals;
```

```
import static org.testng.Assert.assertNotNull;
import static org.testng.Assert.assertTrue;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.ws.client.core.WebServiceTemplate;
import org.springframework.ws.soap.client.SoapFaultClientException;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import io.educative.soap_automation.CreateStudentRequest;
import io.educative.soap_automation.CreateStudentResponse;
import io.educative.soap_automation.DeleteStudentRequest;
import io.educative.soap_automation.DeleteStudentResponse;
import io.educative.soap_automation.GetStudentsRequest;
import io.educative.soap_automation.GetStudentsResponse;
import io.educative.soap_automation.UpdateStudentRequest;
import io.educative.soap_automation.UpdateStudentResponse;

public class TestSOAP extends BaseTest {

        @Test
        public void testCreateUpdateDeleteStudent() {

                // Creating Student with following details
                CreateStudentRequest createStudentRequest = new CreateStudentRequest();
                createStudentRequest.setGender("Female");
                createStudentRequest.setFirstName("Sam");
                createStudentRequest.setLastName("Bailey");

                // Making Create Student web service call
                CreateStudentResponse createStudentResponse = (CreateStudentResponse) webServiceTe
                                SERVICE_URL, createStudentRequest);

                // Validating response is not null
                assertNotNull(createStudentResponse, "CreateStudentResponse is null");

                // Printing the response XML
                printResponse(createStudentResponse);

                // Created Student ID
                long id = createStudentResponse.getStudent().getId();
                GetStudentsRequest getStudentsRequest = new GetStudentsRequest();
                getStudentsRequest.setId(id);

                // Fetching the created Student response
                GetStudentsResponse getStudentsResponse = (GetStudentsResponse) webServiceTemplate
                                SERVICE_URL, getStudentsRequest);

                // Validating response is not null and contains more than one object
                assertNotNull(getStudentsResponse, "GetStudentsResponse is null");
                assertTrue(!getStudentsResponse.getStudents().isEmpty(), "students list is empty")
                assertEquals(getStudentsResponse.getStudents().get(0).getFirstName(), "Sam", "firs

        // Printing the response XML
                printResponse(getStudentsResponse);
```

```
                // Update the Student with information
                UpdateStudentRequest updateStudentRequest = new UpdateStudentRequest();
                updateStudentRequest.setId(id);
                updateStudentRequest.setGender("Male");
                updateStudentRequest.setFirstName("Johnny");
                updateStudentRequest.setLastName("Doe");

                // Making UpdateStudent web service call
                UpdateStudentResponse updateStudentResponse = (UpdateStudentResponse) webServiceTe
                                SERVICE_URL, updateStudentRequest);

                // Validating response not null and the first name as given in the request
                assertNotNull(updateStudentResponse, "UpdateStudentResponse is null");
                assertEquals(updateStudentResponse.getStudent().getFirstName(), "Johnny");

        // Printing the response XML
                printResponse(updateStudentResponse);

                // Fetching the created Student response
                GetStudentsResponse getStudentsResponseAfterUpdate = (GetStudentsResponse) webServ
                                .marshalSendAndReceive(SERVICE_URL, getStudentsRequest);

                assertNotNull(getStudentsResponseAfterUpdate, "GetStudentsResponse is null");
                assertTrue(!getStudentsResponseAfterUpdate.getStudents().isEmpty(), "students list

                // Printing the response XML
                printResponse(getStudentsResponseAfterUpdate);

                // Deleting the Student
                DeleteStudentRequest deleteStudentRequest = new DeleteStudentRequest();
                deleteStudentRequest.setId(id);

                // Making DeleteStudent web service call
                DeleteStudentResponse deleteStudentResponse = (DeleteStudentResponse) webServiceTe
                                SERVICE_URL, deleteStudentRequest);

                // Validating response is not null and isDeleted() true on successful delete
                assertNotNull(deleteStudentResponse, "DeleteStudentResponse is null");
                assertTrue(deleteStudentResponse.isDeleted(), "Student not deleted");

                // Printing the response XML
                printResponse(deleteStudentResponse);

                // Fetching the deleted Student and expecting SoapFaultClientException
                try {
                        webServiceTemplate.marshalSendAndReceive(SERVICE_URL, getStudentsRequest);
                } catch (SoapFaultClientException e) {
                        assertEquals(e.getMessage(), String.format("student with id '%s' not found
                }
        }
}

abstract class BaseTest {

        protected static ApplicationContext CONTEXT;

        protected WebServiceTemplate webServiceTemplate;

        protected static final String SERVICE_URL = "http://ezifyautomationlabs.com:6566/educative

        protected static final Logger LOG = LoggerFactory.getLogger(BaseTest.class);
```

```
        @BeforeSuite
        public void init() {
                if (CONTEXT == null) {
                        CONTEXT = new AnnotationConfigApplicationContext(io.educative.soap.WebServ
                }
        }

        // Initializing WebServiceTemplate
        @BeforeClass
        public void initTemplate() {
                webServiceTemplate = CONTEXT.getBean(WebServiceTemplate.class);
        }

        // Printing XML Response
        protected void printResponse(Object response) {
                try {
                        LOG.info("printing response '{}' => \n{}", response.getClass().getName(),
                                        new XmlMapper().writerWithDefaultPrettyPrinter().writeValu
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        // An utility to create student
        protected long createStudent() {
                CreateStudentRequest request = new CreateStudentRequest();
                request.setGender("Male");
                request.setFirstName("John");
                request.setLastName("Doe");

                CreateStudentResponse response = (CreateStudentResponse) webServiceTemplate.marsha
                                request);

                assertNotNull(response, "CreateStudentResponse is null");

                return response.getStudent().getId();
        }
}
```

Console Output

Understanding the code

Since the code above has enough in-line comments, we will only discuss some important code snippets here.

The web service is hosted at `http://ezifyautomationlabs.com:6566/educative-soap/ws` and saved as `SERVICE_URL` in `BaseTest`.

It uses the `TestNG` library to write the test method.

1. Create a new student

```
// Creating Student with following details
CreateStudentRequest createStudentRequest = new CreateStudentRequest();

createStudentRequest.setGender("Female");
createStudentRequest.setFirstName("Sam");
createStudentRequest.setLastName("Bailey");
```

In this piece of code, we create the `CreateStudentRequest` request object to make the `CreateStudent` web service call.

```
// Making Create Student web service call
CreateStudentResponse createStudentResponse = (CreateStudentResponse) webServi
ceTemplate.marshalSendAndReceive(SERVICE_URL, createStudentRequest);
```

Here, we use `WebServiceTemplate`'s `marshalSendAndReceive` method to make the web service call by passing arguments – `SERVICE_URL` that denotes the location where the service is hosted and the `CreateStudent` request object. The `marshalSendAndReceive` method internally converts the request object to the XML that `SOAP` web service understands, sends it over `HTTP`, receives the response and converts the XML response returned by the web service to a Java object. In this case, the `CreateStudentResponse` object.

2. Validating the created student

After receiving the response, we validate the response for correctness.

Using `TestNG`'s `assertNotNull` method, we check whether the received response is ***Not Null***.

```
// Validating response is not null
assertNotNull(createStudentResponse, "CreateStudentResponse is null");
```

3. Fetching created student information

Once the validation is done, we fetch the `id` of the created `Student` to be used later in the flow.

```
// Created Student ID
long id = createStudentResponse.getStudent().getId();
```

We create the `GetStudentsRequest` request object with the previously created Student `id`.

```
GetStudentsRequest getStudentsRequest = new GetStudentsRequest();
getStudentsRequest.setId(id);
```

We make the `GetStudents` web service call the same as explained before to create `Student`.

```
// Fetching the created Student response
GetStudentsResponse getStudentsResponse = (GetStudentsResponse) webServiceTemp
late.marshalSendAndReceive(SERVICE_URL, getStudentsRequest);
```

Here, we validate the fetched information with the data that we passed while creating `Student`.

```
// Validating response is not null and contains more than one object
assertNotNull(getStudentsResponse, "GetStudentsResponse is null");
assertTrue(!getStudentsResponse.getStudents().isEmpty(), "students list is emp
ty");
assertEquals(getStudentsResponse.getStudents().get(0).getFirstName(), "Sam",
"first name");
```

4. Updating the student

Here we create the `UpdateStudentRequest` request object to call `UpdateStudent` web service for the same `id` that we created earlier.

```
// Update the Student with information
UpdateStudentRequest updateStudentRequest = new UpdateStudentRequest();
updateStudentRequest.setId(id);
updateStudentRequest.setGender("Male");
updateStudentRequest.setFirstName("Johnny");
updateStudentRequest.setLastName("Doe");
```

```
// Making UpdateStudent web service call
UpdateStudentResponse updateStudentResponse = (UpdateStudentResponse) webServi
ceTemplate.marshalSendAndReceive(SERVICE_URL, updateStudentRequest);
```

```
// Validating response not null and the first name as given in the request
assertNotNull(updateStudentResponse, "UpdateStudentResponse is null");
assertEquals(updateStudentResponse.getStudent().getFirstName(), "Johnny");
```

5. Fetching updated student information

As seen before, we call `GetStudents` web service again, to see whether the information of the `Student` identified by the `id` is updated with correct details by

validating the response data. This step is the same as step 3.

```
// Fetching the created Student response
GetStudentsResponse getStudentsResponseAfterUpdate = (GetStudentsResponse) web
ServiceTemplate.marshalSendAndReceive(SERVICE_URL, getStudentsRequest);
```

6. Deleting the student

We create the `DeleteStudentRequest` request object with the `id` we created earlier.

```
// Deleting the Student
DeleteStudentRequest deleteStudentRequest = new DeleteStudentRequest();
deleteStudentRequest.setId(id);
```

We make a `DeleteStudent` web service with the request object and receive `DeleteStudentResponse` as response.

```
// Making DeleteStudent web service call
DeleteStudentResponse deleteStudentResponse = (DeleteStudentResponse) webServi
ceTemplate.marshalSendAndReceive(SERVICE_URL, deleteStudentRequest);
```

Next, we validate the response to see whether `isDeleted` field is set to ***true***.

```
// Validating response is not null and isDeleted() true on successful delete
assertNotNull(deleteStudentResponse, "DeleteStudentResponse is null");
assertTrue(deleteStudentResponse.isDeleted(), "Student not deleted");
```

7. Fetching the deleted student information

In addition to validating `isDeleted` of `DeleteStudentResponse`, we check whether no (any?) information is returned for the `id` that we created earlier. In our case, we expect a `SoapFaultClientException` exception with the message, ***"student with id '${id}' not found"***. This ensures the complete erasure of the Student and no records found for the given `id`.

Note: We highly recommend putting all the web service calling code as functions with appropriate data as parameters in a separate class(es) to have the reusability of methods across different test classes.

We hope this lesson was elaborate enough to help you understand how to make web service calls, chain them to the subsequent web service calls and form an integration test flow. In the next chapter, we will learn about Allure reporting and

integration test flow. In the next chapter, we will learn about Allure reporting and its integration in the tests.