“Navigate the Automation Seas: A Practical Journey with Live Showcases"

Aila Bogasieru

Chapter 1

Introduction

As a Software Engineer with over 10 years of experience, I began my journey in test automation in 2010, tackling the challenges of web applications using Java and Selenium. While there's an abundance of documentation and courses available on platforms like YouTube, Coursera, Udemy, and Pluralsight, few truly demonstrate how to build a reliable and maintainable testing framework.

In my humble opinion, it's more important to focus on the logic behind writing tests than on the specific framework or programming language used, specific framework or programming language but in the underlying logic of writing robust tests. While the choice of technology is indeed vital, it is equally important to architect automation frameworks that are streamlined and efficient, leveraging the capabilities of various libraries.

Keeping it simple is essential. Adding unnecessary complexity can hinder the effectiveness of your automation framework. My technology stack primarily consists of Java, Selenium (for web applications),
Rest Assured (for RESTful APIs), JUnit/TestNg (as testing frameworks), Apache Maven (as the build tool), and CI/CD tools like Jenkins.

Over the past four years, there has been a notable shift toward Python, leading to a decline in Java's popularity. The pandemic further accelerated the demand for AI-driven projects, reshaping automation practices. Concurrently, "low-code" platforms such as Katalon and ReadyAPI have emerged, yet they often fall short in delivering the necessary speed, robustness, and reliability. These platforms exhibit significant limitations, particularly the absence of object-oriented programming (OOP) capabilities, which are crucial for creating scalable and efficient test automation solutions.

In the first years of automation testing, automation frameworks were built from scratch (Selenium was created later on), so every click, list, button, text field needed to be created, so more time and effort was put into the automation testing frameworks. As technology continues to evolve, we witness enhancements in performance, data handling, and user-friendly interfaces for web applications. This evolution extends beyond web applications to encompass desktop and mobile applications, as well as RESTful APIs, broadening the scope of automation from basic frameworks to more scalable and robust solutions.

In the first years of automation testing, automation frameworks were built from scratch (Selenium was created later on), so every click, list, button, text field needed to be created, so more time and effort was put into the automation testing frameworks. Once Selenium gained traction and

popularity because having amazing capabilities to save a lot of hard work on coding, bringing solutions to handle multi browsers, languages among others, it was clearly a step ahead for automation testing.

Even though we are living in a society where technology and speed are playing an important role, just a few clicks and we have our products delivered, streaming platforms available (no need to actually go to a cinema theater), basically we have applications to solve almost any requirement, we should not forget and be aware about the quality of the software products that could be significantly impacted.

Another important factor to be considered in the product's quality might be the shift from office work to remote work and this resulted in cost-efficiency alternatives by hiring people not that skilled.

Chapter 2

Part 1: Introduction to Automation Testing

- **Chapter 1: Understanding the Basics of Automation Testing**
  - ○ Benefits of automation testing
  - ○ Types of automation testing (unit, regression, system, end-to-end and integration)
  - ○ Challenges and misconceptions
- **Chapter 2: Choosing the Right Automation Framework**
  - ○ Popular frameworks (Selenium and Cypress)
  - ○ Factors to consider when selecting a framework
  - ○ Case study of successful framework implementations for web app
- **Chapter 3: API Testing**
  - ○ RESTful APIs automated tests

Chapter 3

Part 2: Practical Automation Techniques

- **Chapter 4: Setting Up Your Automation Environment**
  - ○ Installing necessary tools and libraries
  - ○ Configuring your testing environment
- **Chapter 5: Creating Effective Test Scripts**
  - ○ Writing maintainable and reusable test scripts
  - ○ Best practices for test script design
- **Chapter 6: Continuous Integration and Continuous Delivery (CI/CD)**
  1. ○ Integrating automation testing into CI/CD
  2. ○ Benefits of CI/CD
  3. ○ Best practices for CI/CD implementation
- **Chapter 7: Low-Code Platforms in Practice**
  1. ○ Overview of low-code platforms (Katalon, ReadyAPI)
  2. ○ Key limitations (e.g., lack of OOP, redundancy issues)

Chapter 4

Part 3: Case Studies and Best Practices

- **Chapter 8: Real-World Automation Success Stories**
    1. ○ Case studies of successful automation implementations for web apps, RESTful APIs, Kafka streams and Databases
    2. ○ Lessons learned and best practices

Chapter 5

Part 4: Overcoming Common Challenges

- **Chapter 9: Addressing Common Issues in Automation Testing**
  1. ○ Tips for troubleshooting and debugging

- **Chapter 5: Creating Effective Test Scripts**
  - ○ Writing maintainable and reusable test scripts
  - ○ Best practices for test script design

○ Writing maintainable and reusable test scripts

○ Best practices for test script design

A good testing automation framework can be measured by different metrics:
test coverage, test execution time, test maintenance effort, bugs detection rate, false positive rate, return of investment, scalability, integration with CI/CD, reporting and analytics. Is really important to pay attention to the architecture of the automation solution that we want to build in scope of any types of tests we have: functional, system, integration, end-to-end and regression.

To achieve this we need to write code that is easy to understand and maintain with as little of effort as possible. We can obtain this by using:

1. Design patterns: **Factory**, **Builder**, **Page Object Model**, **Singleton**, **Template**, **Data-Driven Testing**.

Template: in Case 2: dxp-data project we built data modeling for APIs' payloads with methods allowed and urls set in a configuration file we build the request and response message.

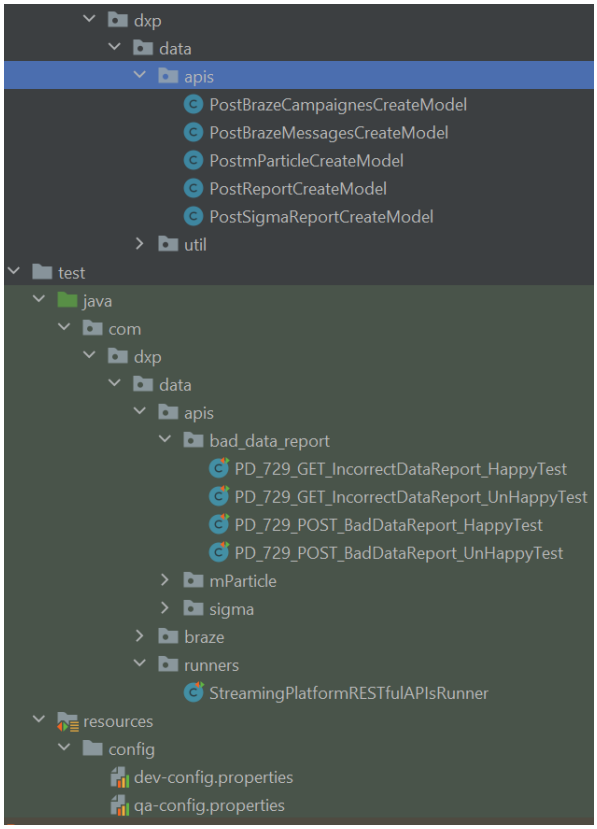

Figure 5.1: dxp-data architecture test

The logic and implementation can be re-checked in **Chapter 3: API Testing, Case 2**.

2. Structure of the tests and test suites: clean, clear names.

2.2 JUnit tests, we have runners to organize and prioritize our tests. Going back to **Chapter 3** to our real-world project: **dxp-data** covering RESTful APIs automated tests, the test suite is defined as a runner containing the list of tests based on the existing tests:

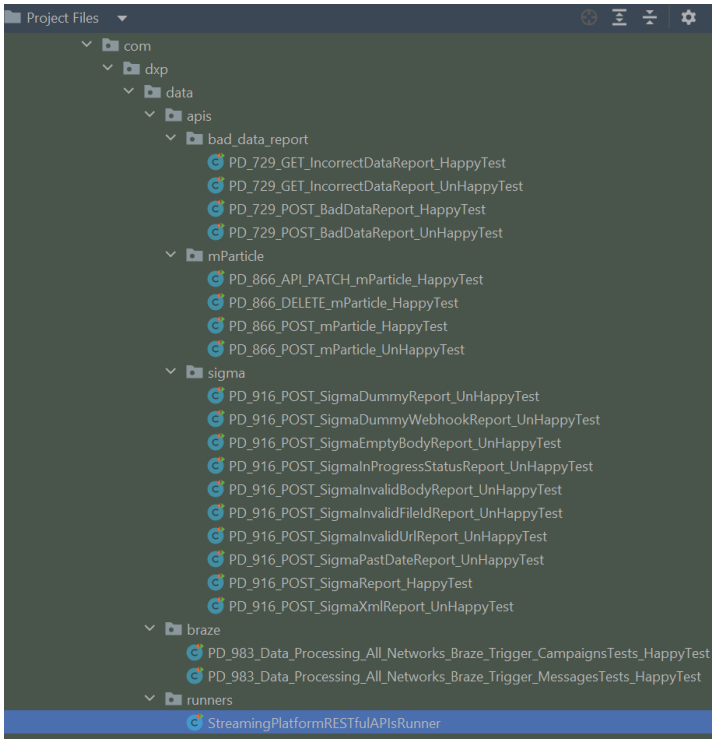

Figure 5.2: tests structure

Setting our tests from Figure 7.9, runner is illustrated below:

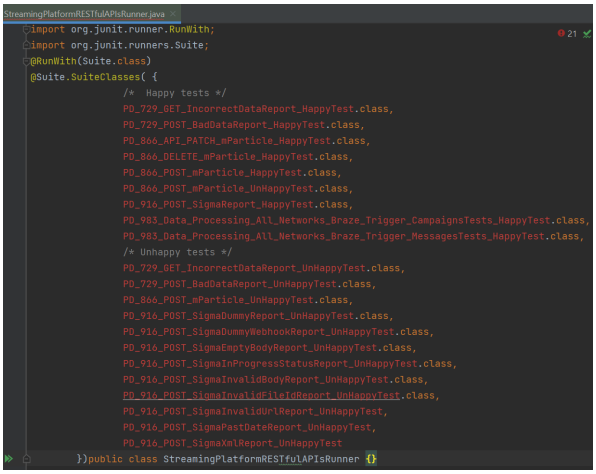

Figure 5.3: dxp data runners errors

Errors are showing, no imports being added for all JUnit classes in use, after all the imports are added runner is runnable.

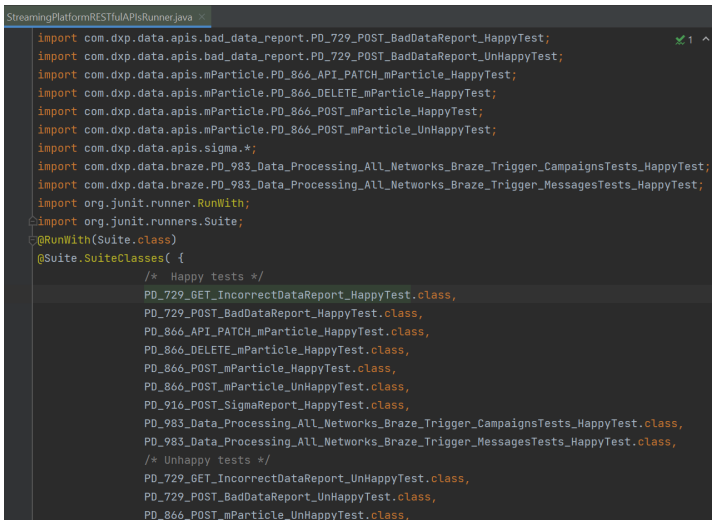

Figure 5.4: dxp data runner successful

3. Custom written tests can be specific by addressing issues and end cases in application reducing the frequency of failures.

By integrating different components and simulating real-world scenarios our tests will provide a more accurate assessment of the system's performance.

Custom tests can be quickly adapted to reflect changes in our application ensuring continuous and up-to-date coverage.

Test autonomy by generating data we are enabling more reliable validations while reducing reliance on potentially outdated, insufficient or non-existent test data.

Diving into the automated solution for RESTful APIs, we can choose test: PD 866 POST mParticle Happy Test, payload is set with dynamic values: data plan id, data plan name, description, event name, on each call is made.

```
public class PD_866_POST_mParticle_HappyTest {

    protected static final Logger logger = LoggerFactory.getLogger(
    PD_866_POST_mParticle_HappyTest.class);

    private RestAssuredConnector connector = new RestAssuredConnector();

    public static final String DATA_PLAN_ID = "auto" + "_"
    + "data_plan" + GenerateRandomDataUtils.generateRandomNumber(4);

    public static final String DATA_PLAN_NAME = "Mobile Data Plan" +
        GenerateRandomDataUtils.generateRandomString(4);

    public static final String DATA_PLAN_DESCRIPTION = "Auto" +
        GenerateRandomDataUtils.generateRandomNumber(2) +
   "This is an example data plan description.";

    public static final Integer VERSION = 1;
    public static final String ACTIVATED_ENVIRONMENT = "none";

    public static final String DESCRIPTION = "Auto" +
        GenerateRandomDataUtils.generateRandomString(4) + "data point";
```

```
public static final String MATCH_TYPE = "custom_event";

public static final String EVENT_NAME = "Auto" +
    GenerateRandomDataUtils.generateRandomString(4) + "My Custom Event
    Name";
public static final String CUSTOM_EVENT_TYPE = "other";
public static final String VALIDATOR_TYPE = "json_schema";
public static final boolean ADDITIONAL_PROPPERTIES = true;
public static final String FOO_TYPE = "number";
private String requestBodymParticle;
private String proxy;
private String requestUrimParticle;

@Before
public void setupData() {

    PostmParticleCreateModel requestMessageBodymParticle = new
        PostmParticleCreateModel();

    requestMessageBodymParticle.setData_plan_id(
    DATA_PLAN_ID);
    requestMessageBodymParticle.setData_plan_name(
    DATA_PLAN_NAME);
    requestMessageBodymParticle.
    setData_plan_description(DATA_PLAN_DESCRIPTION);
    requestMessageBodymParticle.setVersion(VERSION);
    requestMessageBodymParticle.
    setActivated_environment(ACTIVATED_ENVIRONMENT);
    requestMessageBodymParticle.
    setDescription(DESCRIPTION);
    requestMessageBodymParticle.
    setMatchType(MATCH_TYPE);
    requestMessageBodymParticle.
    setEvent_name(EVENT_NAME);
    requestMessageBodymParticle.
    setCustom_event_type(CUSTOM_EVENT_TYPE);
    requestMessageBodymParticle.
```

```
    setValidator_type(VALIDATOR_TYPE);
    requestMessageBodymParticle.
    setAdditionalProperties(ADDITIONAL_PROPPERTIES);
    requestMessageBodymParticle.
    setFoo_type(FOO_TYPE);

    requestBodymParticle = requestMessageBodymParticle.toString();

    requestUrimParticle = ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
    BASE_URL_mParticle.toString())
    +
    ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
        END_URL_mParticle_AMCN_TEST.toString());

    proxy = ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
    PROXY_QA.toString());

}

@Test
public void postmParticleHappyTest() {

    logger.info("mParticle API call create new data plan: ");

    Map<String, String> headermParticle = new HashMap<>();
    headermParticle.put("Content-Type", "application/json");

    Response mParticleResponse =
        connector.postRequest(requestUrimParticle, proxy, headermParticle,
        requestBodymParticle);
    logger.info("requestUrimParticle: " + " " + requestUrimParticle);

    logger.info("requestBodymParticle: " + " " + requestBodymParticle);

    assertEquals(200, mParticleResponse.getStatusCode());
    logger.info("Response: " + mParticleResponse.getBody().asString());
    Assert.assertTrue(mParticleResponse.getBody().
```

```
        asString(), true);

    }
}
```

4. Logging information in our tests. To have more detailed information about the tests' progress, debugging and troubleshooting scope.
By logging important key events, functional calls, progress of the method or test execution you can track and monitor when and why something went wrong.

Logs can capture relevant information that helps recreate issues making the debugging more efficient. Logs can help to understand how it works, valuable information to make better decisions about design, features and enhancements.

Popular logging frameworks **Log4j**, **Logback** and **SLF4J**. Log meaningful information: threads, timestamps, exceptions. Logging is essential, excessive logging can impact performance.
In all code snippets we seen examples of logging, requires sl4j maven dependency, set in pom.xml file:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>${slf4j.version}</version>
 </dependency>
```

necessarily imports in our tests:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

 protected static final Logger logger = LoggerFactory.getLogger(
 POST_SigmaPastDateReport_UnHappyTest.class);

//test logic in here
 logger.info("requestUrimParticle: " + " " + requestUrimParticle);
 logger.info("requestBodymParticle: " + " " + requestBodymParticle);
```

```
logger.info("Response: " + mParticleResponse.getBody().asString());
logger.info("response code: " + reportResponse.getStatusCode());
```

code snippet logs information about: **url**, **payload**, **response message** and **response code** for the API in use.

• **Chapter 7: Low-Code Platforms in Practice**

1. ○ Overview of low-code platforms (Katalon, ReadyAPI)
2. ○ Key limitations (e.g., lack of OOP, redundancy issues)

○ Overview of low-code platforms (Katalon, ReadyAPI, etc.)

1. Katalon Tool. In the last couple of years many companies made a shift from building testing automation solutions using popular programming languages: Java, C and Python to "low-code" platforms, Katalon being one of them.

Its popularity might be related to the fact that it is addressed for people not that skilled in coding and is built to support web, API, mobile and desktop applications. Sounds like a beautiful dream, to have the amount of work almost done, but we should know better that if we have this type of solution there is always a catch or multiple.

Even though supports testing for all types of applications and has integration with CI/CD like: Jenkins, GitLab, Azure DevOps and comes with a variety of features like: Katalon Studio, Katalon TestOps, Katalon Recorder, Katalon Runtime Engine and Katalon TestCloud, Katalon comes with a price: is a licensed software and has a lot of limitations.

In just half a year of using it I did notice a lot of disadvantages. Katalon's interface seems user friendly at first, to me was not that intuitive with hidden fields and functionalities.

In terms of documentation and community support, even though it is a growing community it is not that easy to find information when you're running into problems.

Built-in solutions definitely are more difficult to manage, understand, scale, debug and meet different scope for automated tests. Katalon is a commercial product, so a license needs to be purchased to use it, for companies with limited budgets this can become a great inconvenience.

There are some performance issues, glitches, crashes, non responsive, increased execution time for large-scale test automation projects or complex test scenarios. All of these make us dependent on the tool itself.

2. ReadyAPI Tool.

Aside from Katalon, ReadyAPI is another "low-code" platform emerging as an alternative for no or less coding. ReadyAPI is limited only to RESTful APIs and Kafka streams capabilities. Supports functional testing, performance testing, security testing, mock services, API virtualization, integration with CI/CD through TestEngine.

In one year of using ReadyAPI, I did notice a lot of disadvantages. The same as Katalon to me is not that user friendly, hidden fields and functionalities could require a pretty decent amount of time spent to learn how tests can be set and configured.

The same as Katalon, ReadyAPI is a commercial product and requires purchasing a license to use it.

While writing Groovy test scripts I experienced many crashes, performance issues, increased time execution and almost impossible to do a proper debug. All of these make us dependent on the tool itself.

Other issues and limitations noticed:

- redundant, duplicate code;
- bulk code;
- doesn't support rest assured library;
- no possibility to have classes/methods or OOP;
- no possibility to order tests;
- no possibility to extend to multiple environments;
- no possibility to write multiple tests;
- no possibility to create easily integration tests to validate, track, process and generate data in different systems;
- no stack trace for failed tests;
- difficult git workflow (code review especially)
- not able to solve conflicts easily (really difficult to go through xml files.

Kafka Consumer Groovy test script:

```
package kafka.consumers

import org.apache.kafka.clients.consumer.*

import java.text.SimpleDateFormat

String pattern = "mm-dd-yyyy"
SimpleDateFormat dateFormat = new SimpleDateFormat(pattern)

String currentDate = dateFormat.format(new Date())

String uniqueGroupID = "automation_doc_events_consumer" + currentDate

String bootstrapServers = ""

String topicName = ""

String schemaRegistryUrl = ""
String trustStoreLocation = ""

log.info ("Adding properties to Kafka Consumer")

Properties props = new Properties()
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
props.put(ConsumerConfig.GROUP_ID_CONFIG, uniqueGroupID)
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "");
props.put("security.protocol", "")
props.put("sasl.mechanism", "")
props.put("basic.auth.credentials.source", "")
props.put("basic.auth.user.info", "")
props.put("sasl.jaas.config", "")
props.put("ssl.truststore.location", trustStoreLocation)
props.put("ssl.truststore.password", "changeit")
props.put("schema.registry.url", schemaRegistryUrl)
```

```
props.put("schema.registry.ssl.truststore.location", trustStoreLocation)
props.put("schema.registry.ssl.truststore.password", "changeit")
props.put("auto.offset.reset", "latest")

log.info("Initializing and subscribe to the topic")

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props)
consumer.subscribe(Arrays.asList(topicName))
consumer.close()

log.info("Kafka consumer SetUp... with success")
```

RESTful API Groovy test script example:

```
package restful_apis.post_get

import okhttp3.*
import okhttp3.MediaType
import org.json.JSONArray
import org.json.JSONObject
import java.util.Random
import java.text.DateFormat
import java.text.SimpleDateFormat
import org.assertj.core.api.SoftAssertions

String base_receive = ""
String middle_receive = ""
String end_url_receive = ""

Random random = new Random()
int randomInt = random.nextInt(10000000)
String key = randomInt.toString()

Date now = new Date()
SimpleDateFormat date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
String currentDateComplete = date.format(now)
```

```
String url_receive = base_url_receive + middle_url_receive
+ end_url_receive

String username = ""
String password = ""

log.info("Creating POST Receive Call ..." + url_receive)

JSONObject requestBodyJson = new JSONObject()
requestBodyJson.put("key", key)
requestBodyJson.put("id", "")
requestBodyJson.put("amount", 500000)
requestBodyJson.put("quoteDate", currentDateComplete)

RequestBody receiveBody =
    RequestBody.create(MediaType.parse("application/json"),
    requestBodyJson.toString())

OkHttpClient client = new OkHttpClient()
Request request_receive = new
    Request.Builder().url(url_receive.header("Authorization",
    Credentials.basic(username, password)).header("Content-Type",
    "application/json").post(receiveBody).build()
log.info("Receive Call:" + request_receive)

Response response_receive = client.newCall(request_receive).execute()
String responseBody_receive= response_receive.body().string()

log.info("response:" + responseBody_receive)
int statusCode_receive = response_receive.code()

SoftAssertions softly = new SoftAssertions()

softly.assertThat(statusCode_receive).isEqualTo(200)
softly.assertAll()
```

```
log.info("Response for Receive Call :" + responseBody_receive)
log.info("Status code for Receive Call:" + statusCode_receive)

//Save key
JSONObject jsonObject = new JSONObject(responseBody_receive)
JSONArray msgArray = jsonObject.getJSONArray("messages")
JSONObject msgObject = msgArray.getJSONObject(0)
String msgDesc = msgObject.getString("msgDesc")
String storedKeyString = msgDesc.substring(msgDesc.lastIndexOf(" ") + 1)
int storedKey = Integer.parseInt(storedKeyString.trim())

//GET Receive
String url_receive_get = url_receive + key
log.info("Creating GET Receive with ..." + url_receive_get)

Request request_receive_get = new Request.Builder().url(url_receive_get).
header("Authorization", Credentials.basic(username,
    password)).header("Content-Type", "application/json").build()

log.info("Calling the GET Receive ...")

Response response_receive_get = client.newCall(request_receive_get).execute()
String responseBody_receives_get= response_receive_get.body().string()
int statusCode_receive_get = response_receives.code()

softly.assertThat(statusCode_receive).isEqualTo(200)
softly.assertAll()

log.info("Response for GET Receive :" + responseBody_receive_get)
log.info("Status code for GET Receive :" + statusCode_receive_get)

//DELETE Receive
String url_receive_delete = url_receive + storedKey
log.info("Creating POST Receive Call ..." + url_receive)

Request request_receive_delete = new
    Request.Builder().url(url_receives_delete).
```

```
header("Authorization",Credentials.basic(username, password))
.header("Content-Type", "application/json").delete().build()
log.info("Receive Call:" + request_receive_delete)

Response responsey_receive_delete =
    client.newCall(request_receive_delete).execute()
String responseBody_receive_delete= response_receive_delete.body().string()
int statusCode_receive_delete = response_receive_delete.code()

softly.assertThat(statusCode_receive).isEqualTo(200)
softly.assertAll()

log.info("Response for DELETE Receive Call :" + responseBody_receive)
log.info("Status code for DELETE Receive Call:" + statusCode_receive)
```

Let's analyze what is wrong with the code:

- a lot of duplicate, redundant code due to limitation of methods definition, encapsulation and inheritance;
- difficult to read, understand and follow the steps and the data flow in the test;
- extensive logging (for a better code reading);
- adding more complexity of the test implies an extra level of confusion and difficulty in understanding the test (let's think about the following use case: publish a message produced by Kafka Producer, consume the message with Kafka Consumer, make a RESTful call and make DB data validations);
- old library is used: okhttp3.* (doesn't support rest assured library);
- every time we need to make request calls for the RESTful APIS, the same code is rewritten;
- JSON Objects for payload and response messages need to be rewritten every time it is used without the data modeling for the payload.

Let's see how it looks the test implementation for the steps described:

1. Publish message on a topic produced with Kafka Producer;

2. Consume the message with Kafka Consumer;
3. Make necessarily RESTful APIs call and response message data validations and store data;
4. Make data validation in DB.

```
package kafka.producer_micros_consumer_db

import java.text.SimpleDateFormat

import org.apache.kafka.clients.producer.KafkaProducer
import org.apache.kafka.clients.producer.ProducerConfig
import org.apache.kafka.clients.producer.ProducerRecord

import org.apache.avro.Schema
import org.apache.avro.generic.GenericData
import org.apache.avro.generic.GenericRecord
import org.apache.kafka.clients.producer.RecordMetadata

import org.apache.kafka.clients.consumer.*
import org.apache.kafka.clients.consumer.KafkaConsumer
import org.apache.kafka.clients.consumer.ConsumerRecords
import org.json.JSONArray
import org.json.JSONObject

import java.sql.Connection
import java.sql.PreparedStatement
import java.sql.DriverManager
import java.sql.ResultSet
import org.assertj.core.api.SoftAssertions

//Producer variables
String bootstrapServers = ""

String topicProducer = ""

String trustStoreLocation = ""
String schemaRegistryUrl = ""
```

```
Date now = new Date()
SimpleDateFormat date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
String currentDateComplete = date.format(now)

UUID generateUUID = UUID.randomUUID()

SimpleDateFormat inputFormat = new SimpleDateFormat
("E MMM dd HH:mm:ss z yyyy")
inputFormat.setTimeZone(TimeZone.getTimeZone("EDT"))
String dateCreated = inputFormat.format(now)

String eventType= ""
String eventSubtype = ""
String eventId = generateUUID.toString()
String evntSrcDesc = "Dummy desc"
//define all necessary variables

String GUID = '{' + generateUUID.toString() + '}'

//Kafka Consumer variables to setup the props: groupId, brokers, topic, schema
String uniqueGroupID = "automation_events_consumer" + currentDateComplete
String topicConsumer = ""

//RESTful APIs calls
//Define necessary variables for RESTful ASPIs in use
String base_url_agr_c = ""
String middle_url_agr_c = ""
String end_url_agr_c_agrKey = ""
String url_agrKey = base_url_agr_c + middle_url_agr_c + end_url_agr_c_agrKey

String base_url_c = ""
String middle_url_c = ""

String expectedRole = ""

String username_agr = ""
```

```
String password_agr = ""
String username_c = ""
String password_c = ""

//MySQL variables
String connectionUrl = ""
String username = ""
String password = ""
String queryMessage = ''
String appId = ""
String msgType = ""

log.info("Adding properties to Kafka Producer")

Properties propsProducer = new Properties()

propsProducer.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
propsProducer.put(ProducerConfig
.KEY_SERIALIZER_CLASS_CONFIG,
org.apache.kafka.common.serialization.
StringSerializer.class);
propsProducer.put(ProducerConfig.
VALUE_SERIALIZER_CLASS_CONFIG,
io.confluent.kafka.serializers.KafkaAvroSerializer.class);
propsProducer.put("security.protocol", "")
propsProducer.put("sasl.mechanism", "")
propsProducer.put("basic.auth.credentials.source", "")
propsProducer.put("basic.auth.user.info", "")
//add Kafka Producer props

log.info("Producing valid System message on topic: " + topicProducer)

String avroSchema = '{}' //add Avro schema to validate against
Schema.Parser parser = new Schema.Parser()
Schema avroSchemaSystem = parser.parse(avroSchema)

GenericRecord avroRecord = new GenericData.Record(avroSchemaSystem)
```

```
// log.info(avroRecord.toString())

GenericRecord eventHeader = new
    GenericData.Record(avroSchemaSystem.getField("eventheader").
schema())

eventHeader.put("eventType", eventType)
eventHeader.put("eventSubtype", eventSubtype)
eventHeader.put("eventDateTime", currentDateComplete)
eventHeader.put("eventGeneratedDateTime", currentDateComplete)
//set all eventHeader fields and values

GenericRecord eventBody = new GenericData.Record(avroSchemaSystem.
getField("eventBody").schema())

eventBody.put("pfx", pfx)
eventBody.put("objS", objS)
eventBody.put("MimeType", mimeType)
eventBody.put("DateCreated", dateCreated)
//add eventBody values


//propertiesList implementation
List<GenericRecord> propertiesList = new ArrayList<>()

Schema propertiesListSchema = avroSchemaSystem.getField("eventBody").schema().
getField("propertiesList").schema().

getElementType()
GenericRecord propertyRecord = new GenericData.Record(propertiesListSchema)

propertyRecord.put("name", "PropertyName")
propertyRecord.put("value", "PropertyValue")
propertyRecord.put("type", "PropertyType")
//add propertyRecord values
propertiesList.add(propertyRecord)
eventBody.put("propertiesList", propertiesList)
```

```
avroRecord.put("eventheader", eventHeader)
avroRecord.put("eventBody", eventBody)

log.info("System message produced by Kafka Producer: " +
    eventHeader.toString() + eventBody.toString())

ProducerRecord<Object, Object> recordProducer = new
    ProducerRecord<>(topicProducer, null, avroRecord)

KafkaProducer producer = new KafkaProducer(propsProducer)

RecordMetadata metadata = producer.send(recordProducer).get()

log.info("Producer record with eventId: " + eventId.toString())

producer.flush()
producer.close()

log.info("Message sent successfully to topic" + metadata.topic() + " " +
       "partition: " + metadata.partition() + " "+ "offset: " +
           metadata.offset())

log.info("Kafka Producer Prim Id is:" + eventBody.get("prim_id"))
log.info("Sys is:" + eventBody.get("sys"))
String agrKeyProducer = (eventBody.get("prim_id") +
    eventBody.get("sys")).toString()

// Kafka Consumer Setup
log.info ("Adding properties to Kafka Consumer")

Properties propsConsumer = new Properties()
propsConsumer.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
propsConsumer.put(ConsumerConfig.GROUP_ID_CONFIG, uniqueGroupID)
propsConsumer.put(ConsumerConfig.
KEY_DESERIALIZER_CLASS_CONFIG, "");
propsConsumer.put(ConsumerConfig.
```

```
VALUE_DESERIALIZER_CLASS_CONFIG, "");
propsConsumer.put("security.protocol", "")
propsConsumer.put("sasl.mechanism", "")
propsConsumer.put("basic.auth.credentials.source", "")
propsConsumer.put("basic.auth.user.info", "")
//set all the props on Kafka Consumer

log.info("Initializing and subscribing to the topic")

SoftAssertions softly = new SoftAssertions()

KafkaConsumer<String, GenericRecord> consumer = new
    KafkaConsumer<>(propsConsumer)
consumer.subscribe(Arrays.asList(topicConsumer))

ConsumerRecords<String, GenericRecord> records = consumer.poll(1000)

for(ConsumerRecords<String, GenericRecord> record : records) {
    GenericRecord avroRecordConsum = record.value()
    log.info("Current record value: " + record.value().toString())
    String eventIdConsumer =
        avroRecordConsum.get("eventHeader.eventId").toString()

      if(eventId.equals(eventIdConsumer)) {
        softly.assertThat(avroRecordConsum.
        get("eventHeader.eventType").toString()).
        IsEqualTo(eventType)
        softly.assertThat(avroRecordConsum.
        get("eventHeader.eventSubtype").

        softly.assertThat(avroRecordConsum.
        get("eventHeader.eventDateTime").
        toString()).isEqualTo(currentDateComplete)
        softly.assertThat(avroRecordConsum.
        get("eventHeader.
        eventGeneratedDateTime").toString()).
        isEqualTo(currentDateComplete)
```

```
        softly.assertThat(avroRecordConsum.
        get("eventHeader.eventId").
        toString()).isEqualTo(eventId)

        softly.assertThat(avroRecordConsum.
        get("eventHeader.eventRequestId").
        toString()).isEqualTo(eventId)

        softly.assertThat(avroRecordConsum.
        get("eventHeader.evntSrcDesc").
        toString()).isEqualTo(evntSrcDesc)

        //add assertions for all fields
    }
    long offset = record.offset()
    log.info("Consumed record at offset: " + offset)
}
softly.assertAll()
consumer.close()

log.info("Validating the Consumer fields against the fields produced... with
    success, messages is sent on Consumer side")

log.info("Creating Agr Cst with ..." + end_url_agr_c_agrKey)

OkHttpClient client = new OkHttpClient()
Request request_agrKey = new Request.Builder().url(url_agrKey).
header("Authorization", Credentials.basic(username_agr, password_agr)).
header("Content-Type", "application/json").build()

log.info("Agr c MS:" + request_agrKey)

log.info("Calling the Agr Cst...")
Response response_agrKey = client.newCall(request_agrKey).execute()
String responseBody_agrKey = response_agrKey.body().string()
int statusCode_agrKey = response_agrKey.code()
```

```
log.info("Response for agrKey:" + responseBody_agrKey.toString())
log.info("Status code for agrKey:" + statusCode_agrKey)

JSONObject jsonRspAgrC = new JSONObject(responseBody_agrKey)

JSONArray agr = jsonRspAgrC.getJSONArray("agr")
JSONObject c = agr.getJSONObject(0).getJSONArray("agrC").
getJSONObject(0)

JSONArray keyArray = jsonRspAgrC.getJSONArray("agr")
String keyMS = keyArray.getJSONObject(0).getString("agrKey")

String type = c.getString("type")
String mbrGUID_A_C = c.getString("mbrGUID")
log.info("Type role from Agr Cst is:" + type)

//Save mbrGUID
log.info("Saving mbrGUID..")

String url_c = base_url_c + middle_url_c + mbrGUID_A_C
log.info("Creating MS with ..." + url_c)

Request request_cId = new
    Request.Builder().url(url_c).header("Authorization",
    Credentials.basic(username_c, password_c)).
header("Content-Type", "application/json").build()
log.info(" Cst Id:" + request_cId)

log.info("Calling the Cst Id...")
Response response_cId = client.newCall(request_cId).
execute()
String responseBody_customeId= response_cId.body().
string()
int statusCode_cId = response_cId.code()

JSONObject jsonResponsecId = new JSONObject(responseBody_customeId)
```

```
JSONArray cs = jsonResponsecId.getJSONArray("cs")
String mbrGUIDId = cs.getJSONObject(0).
getString("mbrGUID")

if(roleType.equals(expectedRole) && (mbrGUID_A_C).equals(mbrGUIDId)) {
    log.info ("key from producer is: " + agrKeyProducer)
    log.info("key from Agr Cst is: " + keyMS)
}

//DB validations
Connection conn = DriverManager.
getConnection(connectionUrl, username, password)
PreparedStatement statement = conn.prepareStatement(queryMessage)
statement.setString(1, eventId)

ResultSet rs = statement.executeQuery()
if(rs.next()){
    String payload = rs.getString("payload")
    String source = rs.getString("source")
    String type = rs.getString("type")

    JSONObject jsonPayload = new JSONObject(payload)
    JSONObject data = jsonPayload.
    getJSONObject("content").getJSONObject("data")

    String origEvntId = data.getString("origEvntId")
    String origEvntSrcDesc = data.getString("origEvntSrcDesc")
    String origAppId = data.getString("origAppId")
    String number = data.getString("number")
    String origEvntId = data.getString("origEvntId")
    String origEvntDateTime = data.getString("origEvntDateTime")
    JSONArray onsuccess = jsonPayload.getJSONObject("triggers").
    getJSONArray("onsuccess")
    String DBmsgType = jsonPayload.getString("msgType")

    if (origEvntId.equals(eventId)) {
```

```
        softly.assertThat(origAppId).isEqualTo(appId)
        softly.assertThat(type).isEqualTo("auto")
        //softly.assertThat(origEvntSrcDesc).
        isEqualTo(evntSrcDesc)
        //needs to be fixed Producer: evntSrcDesc = "Dummy description" vs.
            DB: "origEvntSrcDesc":"System notifications"
        softly.assertThat(number).isEqualTo(prim_id)
        log.info("DB policy No:" + number)
        softly.assertThat(origEvntId).isEqualTo(eventId)
        softly.assertThat(origEvntDateTime).
        isEqualTo(currentDateComplete)
        softly.assertThat(DBmsgType).isEqualTo(msgType)
        softly.assertAll()
    }
}
conn.close()

log.info("Validating the db fields against the fields produced... with
    success")
```

A lot of logging is required in order to have code more readable, important fact is that as we add more complexity to our tests and add more steps the bulk code become harder and harder to be followed and this results in difficulties in debugging too.

Based on all the aspects and test scripts mentioned above, writing Ready API test scripts becomes extremely difficult and challenging and impossible for good coding standards and practices to be followed.

Chapter 6

Part 3: Case Studies and Best Practices

• **Chapter 8: Real-World Automation Success Stories**

1. ○ Case studies of successful automation implementations for web apps, RESTful APIs, Kafka streams and Databases
2. ○ Lessons learned and best practices

○ Case studies of successful automation implementation for web apps, RESTful APIs, Kafka streams and Databases.

1. Web apps cases: in **Chapter 2** we have covered step by step guideline on how to create testing automation solutions using Java and Selenium following Page Object Model design pattern in scope of **functional**, **integration**, **end-to-end**, and **regression** testing.

2. Backend systems: RESTful APIs, Kafka streams and DBs.

In our practical projects from previous chapters we covered automating testing solutions for RESTFul APIs in **Chapter 3** in scope of functional and regression testing. Also in **Chapter 7** we covered integration and end-to-end testing with ReadyAPI test scripts for RESTful APIs, Kafka and Database.

Next we are going through the implementation for a testing automation solution covering Kafka Producer/Consumer, RESTful APIs and Databases projects implemented in the same solution.

Case 1: Kafka Producer/Consumer, RESTFul APIs and MySQL database systems.

Data flows between Kafka Producer, Kafka Consumer, RESTful APIs and Database systems.

1. Kafka Producer/Consumer logic implementation is covered using:

- properties set on Producer/Consumer side;
- define Avro schema on Producer so the message is published on a topic;
- subscribe to a Consumer topic;
- consume and validate the message against Avro schema from step 1;

2. RESTful API logic implementation is covered using:
    1. data modeling by creating payloads for POST methods;
    2. creating connector setting POST, PATCH, PUT, GET, DELETE methods and the headers setup;
    3. response message parsing for validations and data stored for next processing steps.
3. Database logic implementation is covered using:
    1. define connection to the database;
    2. define necessarily queries;
    3. interrogate the data and make validations required to follow the same data from the message producer.

Structure of the project is illustrated bellow:

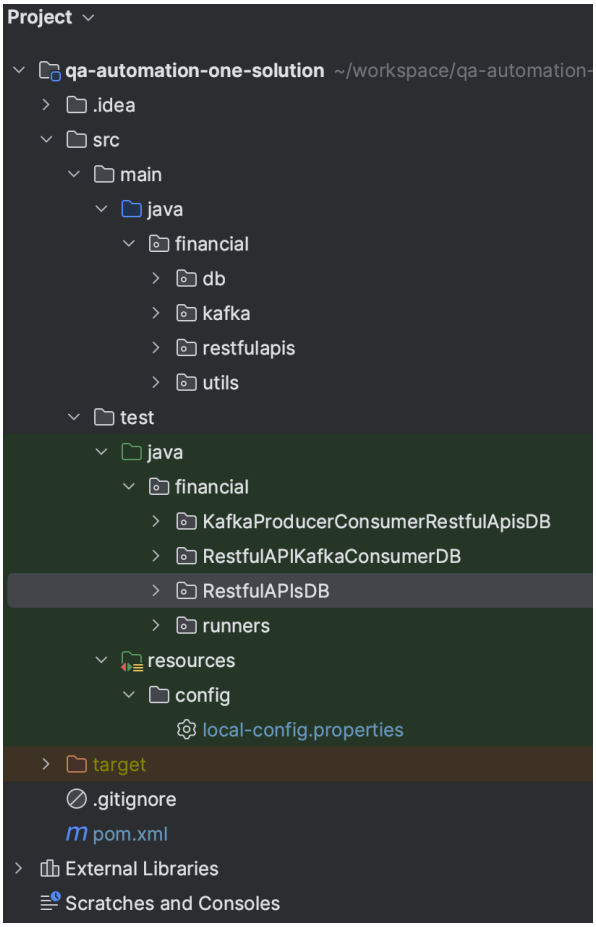

Figure 6.1: Kafka-RESTful APIs-DB project architecture

main package contains necessarily java utilities, setup, data modeling used in the tests' logic.

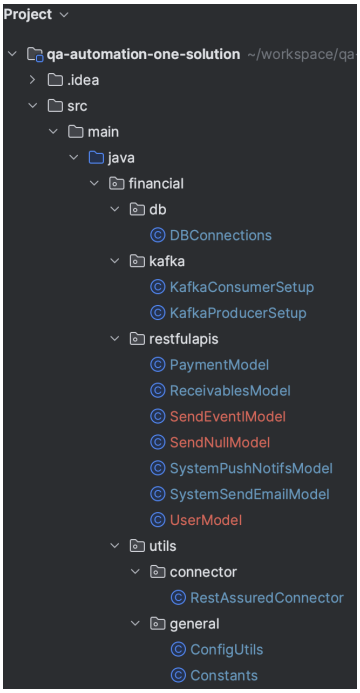

Figure 6.2: Kafka-RESTful APIs-DB main architecture

test package contains a collection of JUnit tests grouped by project packages to cover functional, integration, end-to-end and regression testing.

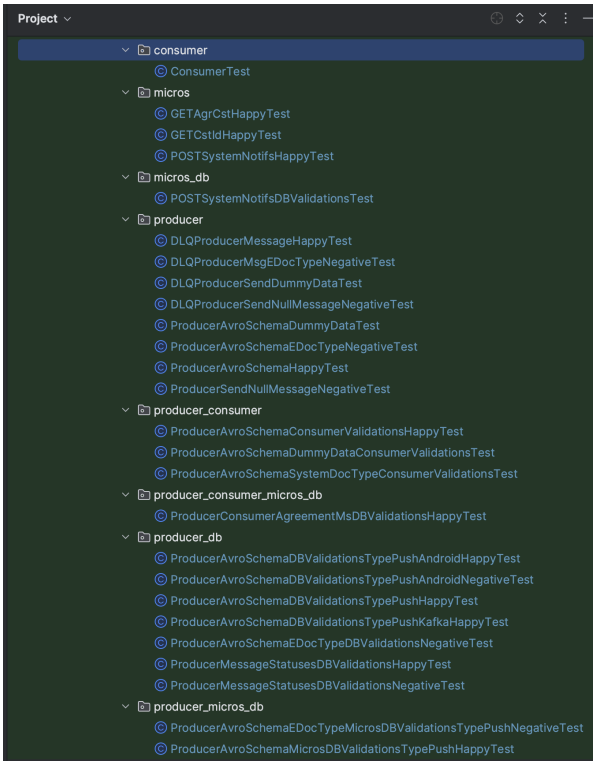

Figure 6.3: Kafka-RESTful APIs-DB test architecture

1.a) **Kafka Consumer** properties set up:

```
package financial.kafka;

import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.serialization.StringDeserializer;

import java.util.Properties;

public class KafkaConsumerSetup {
    public Properties setConsumerProps(String bootstrapServers, String
        uniqueGroupID, String authInfo, String jaasConfig, String
        trustStoreLocation, String schemaRegistryUrl) {
```

```
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, uniqueGroupID);
        props.put(ConsumerConfig.
        KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put(ConsumerConfig.
        VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put("security.protocol", "SASL_SSL");
        props.put("sasl.mechanism", "PLAIN");
        props.put("basic.auth.credentials.source", "USER_INFO");
        props.put("basic.auth.user.info", authInfo);
        props.put("sasl.jaas.config", jaasConfig);
        props.put("ssl.truststore.location", trustStoreLocation);
        props.put("ssl.truststore.password", "changeit");
        props.put("schema.registry.url", schemaRegistryUrl);
        props.put("schema.registry.ssl.truststore.location",
            trustStoreLocation);
        props.put("schema.registry.ssl.truststore.password", "changeit");
        props.put("auto.offset.reset", "latest");
        return props;
    }
}
```

JUnit Test for the **Kafka Consumer** subscribes successfully to a Consumer topic defined in **local-config.properties** file steps:

- Set properties for Kafka Consumer;
- Subscribe to Consumer topic.

```
package financial.KafkaProducerConsumerRestfulApisDB.consumer;

import financial.kafka.KafkaConsumerSetup;
import financial.utils.general.ConfigUtils;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;
import java.util.Properties;

public class ConsumerTest {
    protected static final Logger logger =
        LoggerFactory.getLogger(ConsumerTest.class);
    String pattern = "mm-dd-yyyy";
    SimpleDateFormat dateFormat = new SimpleDateFormat(pattern);
    String currentDate = dateFormat.format(new Date());
    String UNIQUE_GROUP_ID = "automation_events_consumer" + currentDate;

    @Test
    public void subscribeToConsumerTopicHappyTest() {
        KafkaConsumerSetup kafkaConsumerSetup = new KafkaConsumerSetup();
        Properties props = kafkaConsumerSetup.setConsumerProps(ConfigUtils.
        getProperty(String.valueOf(ConfigUtils.
        ConfigKeys.BOOTSTREP_SERVER_QA)), UNIQUE_GROUP_ID,
        ConfigUtils.getProperty(String.valueOf(
        ConfigUtils.ConfigKeys.AUTH_INFO_SYSTEM)),
            ConfigUtils.getProperty(String.valueOf(
        ConfigUtils.ConfigKeys.SASL_CONFIG_SYSTEM)),
            ConfigUtils.getProperty(String.valueOf(
        ConfigUtils.ConfigKeys.TRUST_STORE_LOCATION)),
            ConfigUtils.getProperty(String.valueOf(
        ConfigUtils.ConfigKeys.SCHEMA_REGISTRY)));
        logger.info("Adding properties to Kafka Consumer...");
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList(
        ConfigUtils.getProperty(
        String.valueOf(ConfigUtils.ConfigKeys.
        TOPIC_FINANCE_CONSUMER))));
        consumer.close();
    }
}
```

Kafka Consumer logging:

```
[main] INFO financial.KafkaProducerConsumerRestfulApisDB.consumer.
ConsumerTest - Adding properties to Kafka Consumer...
[main] INFO org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig
    values:
   allow.auto.create.topics = true
   auto.commit.interval.ms = 5000
   auto.include.jmx.reporter = true
   auto.offset.reset = latest
   client.id = consumer-automation_doc_evnt_consumer32-01-2024-1

[main] INFO org.apache.kafka.common.telemetry.internals.
KafkaMetricsCollector - initializing Kafka metrics collector

[main] INFO org.apache.kafka.common.security.authenticator.
AbstractLogin - Successfully logged in.

[main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka version: 3.7.0

[main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka commitId:
    2ae524ed625438c5

[main] INFO org.apache.kafka.common.utils.AppInfoParser - Kafka startTimeMs:
    1727800337389

[main] INFO org.apache.kafka.clients.consumer.internals.
LegacyKafkaConsumer - [Consumer
    clientId=consumer-automation_events_consumer32-01-2024-1,
    groupId=automation_events_consumer32-01-2024] Subscribed to topic(s):
topic_finance

[main] INFO org.apache.kafka.clients.consumer.internals.
ConsumerCoordinator - [Consumer
    clientId=consumer-automation_events_consumer32-01-2024-1,
    groupId=automation_events_consumer32-01-2024] Resetting generation and
    member id due to: consumer pro-actively leaving the group
```

```
[main] INFO org.apache.kafka.clients.consumer.internals.
ConsumerCoordinator - [Consumer
    clientId=consumer-automation_events_consumer32-01-2024-1,
    groupId=automation_events_consumer32-01-2024] Request joining group due
    to: consumer pro-actively
leaving the group

[main] INFO org.apache.kafka.common.metrics.Metrics - Metrics scheduler closed
[main] INFO org.apache.kafka.common.metrics.Metrics - Closing reporter
    org.apache.kafka.common.metrics.JmxReporter
[main] INFO org.apache.kafka.common.metrics.Metrics - Closing reporter
    org.apache.kafka.common.telemetry.internals.
ClientTelemetryReporter
[main] INFO org.apache.kafka.common.metrics.Metrics - Metrics reporters closed
[main] INFO org.apache.kafka.common.utils.AppInfoParser - App info
    kafka.consumer for consumer-automation_events_consumer32-01-2024-1
    unregistered

Process finished with exit code 0
}
```

Consumer Test Overview: logs test result with success:

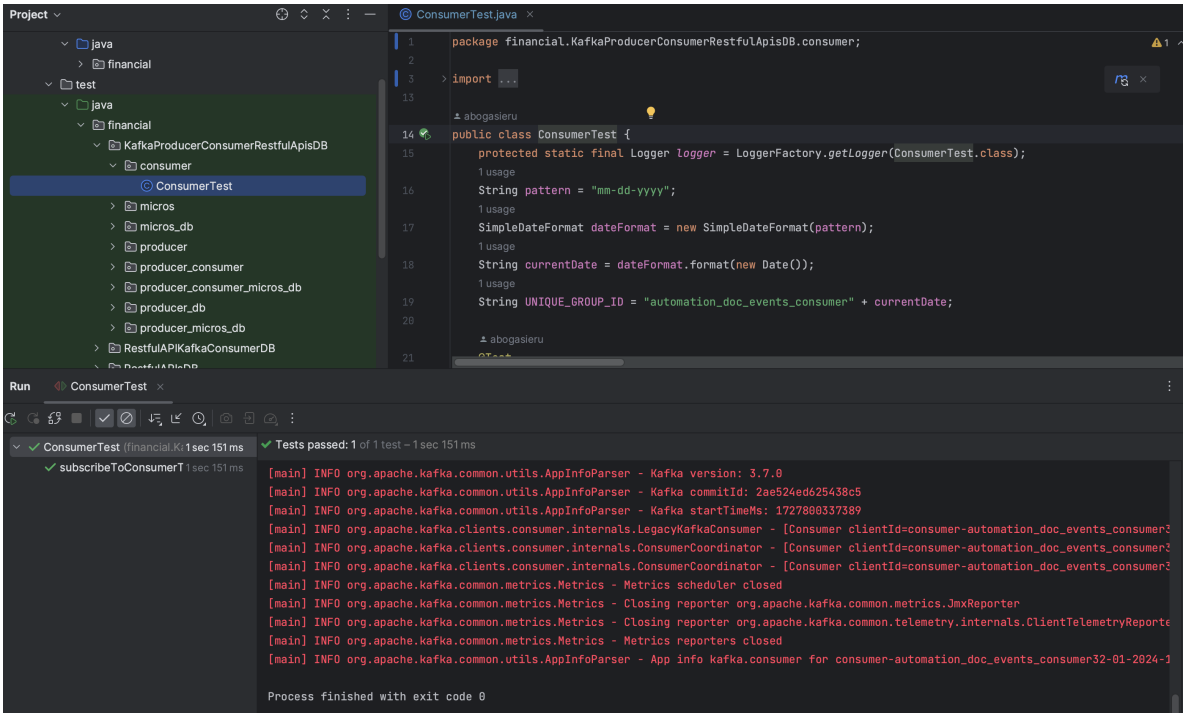

Figure 6.4: Consumer Test Overview

1.b) **Kafka Producer** properties set up:

```
package financial.kafka;

import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
```

```
import org.apache.kafka.clients.producer.ProducerConfig;
import java.util.*;

public class KafkaProducerSetup {

    public Properties setProducerProps(String bootstrapServers, String
        authInfo, String jaasConfig, String trustStoreLocation, String
        schemaRegistryUrl)
    {
        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ProducerConfig.
        KEY_SERIALIZER_CLASS_CONFIG,
        org.apache.kafka.common.serialization.
        StringSerializer.class);

        props.put(ProducerConfig.
        VALUE_SERIALIZER_CLASS_CONFIG,
        io.confluent.kafka.serializers.KafkaAvroSerializer.
        class);
        props.put("security.protocol", "SASL_SSL");
        props.put("sasl.mechanism", "PLAIN");
        props.put("basic.auth.credentials.source", "USER_INFO");
        props.put("basic.auth.user.info", authInfo);
        props.put("sasl.jaas.config", jaasConfig);
        props.put("ssl.truststore.location", trustStoreLocation);
        props.put("ssl.truststore.password", "changeit");
        props.put("schema.registry.url", schemaRegistryUrl);
        props.put("schema.registry.ssl.truststore.
        location", trustStoreLocation);
        props.put("schema.registry.ssl.truststore.password", "changeit");
        props.put("auto.register.schemas", false);
        props.put("specific.avro.reader", true);
        return props;
    }

    public static GenericRecord produceMessage(String eventType, String
```

```
    eventSubtype, String currentDateComplete, String eventId, String
    evnt_desc, String evntS, String d_type, String b_area, String pfx,
String obj_s, String mimeType, String sys_id, String doc, String GUID) {

String avroSchema = "{}"
Schema.Parser parser = new Schema.Parser();
Schema avroSchemaSystem = parser.parse(avroSchema);

GenericRecord avroRecord = new GenericData.Record(avroSchemaSystem);

GenericRecord eventHeader = new GenericData.Record(avroSchemaSystem.
getField("eventheader").
schema());
eventHeader.put("eventType", eventType);
eventHeader.put("eventSubtype", eventSubtype);
eventHeader.put("eventDateTime", currentDateComplete);
eventHeader.put("eventGeneratedDateTime", currentDateComplete);
eventHeader.put("eventId", eventId);
eventHeader.put("evnt_req_id", eventId);
eventHeader.put("evnt_desc", evnt_desc);
eventHeader.put("evntS", evntS);
eventHeader.put("metadata", new HashMap<>());

GenericRecord eventBody = new GenericData.Record(avroSchemaSystem.
getField("eventBody").schema());
eventBody.put("d_type", d_type);
eventBody.put("b_area", b_area);
eventBody.put("pfx", pfx);
eventBody.put("obj_s", obj_s);
eventBody.put("MimeType", mimeType);

eventBody.put("sys_id", sys_id);
eventBody.put("tran_id", new GenericData.Array<>(avroSchemaSystem.
getField("eventBody").schema().
getField("tran_id").schema(),
Arrays.asList("1111")));
```

```
    eventBody.put("DateCreated", currentDateComplete);
    eventBody.put("doc", doc);
    eventBody.put("GUID", GUID);
    eventBody.put("MajorVersion", "1");
    eventBody.put("MinorVersion", "0");

    //propertiesList implementation
    List<GenericRecord> propertiesList =
    new ArrayList<>();

    Schema propertiesListSchema =
        avroSchemaSystem.getField("eventBody").schema().
    getField("propertiesList").schema().
    getElementType();
    GenericRecord propertyRecord = new
        GenericData.Record(propertiesListSchema);

    propertyRecord.put("name", "PropertyName");
    propertyRecord.put("value", "PropertyValue");
    propertyRecord.put("type", "PropertyType");
    propertyRecord.put("multiValue", "PropertyMultiValue");
    propertyRecord.put("multiList", new
        GenericData.Array<>(propertiesListSchema.
    getField("multiList").schema(),
    Arrays.asList("Value1", "Value2")));
    propertiesList.add(propertyRecord);
    eventBody.put("propertiesList", propertiesList);

    avroRecord.put("eventheader", eventHeader);
    avroRecord.put("eventBody", eventBody);

    System.out.println("System message produced: " + eventHeader + eventBody);
    return avroRecord;

}
```

Kafka Producer Test implementation steps:

- Set properties for Kafka Producer;
- Generate a valid Avro schema message;
- Message is successfully published on the partition and offset topic Producer.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.producer;

import financial.kafka.KafkaProducerSetup;
import financial.utils.general.ConfigUtils;
import org.apache.avro.generic.GenericRecord;
import org.junit.Test;
import org.slf4j.*;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.ExecutionException;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class ProducerAvroSchemaHappyTest {
    protected static final Logger logger =
        LoggerFactory.getLogger(ProducerAvroSchemaHappyTest.
    class);

    Date now = new Date();
    SimpleDateFormat date = new SimpleDateFormat(
    "yyyy-MM-dd HH:mm:ss");
    String currentDateComplete = date.format(now);
    UUID generateUUID = UUID.randomUUID();
    String eventType = "eventType";
    String eventSubtype = "eventSubtype";
    String evntId = generateUUID.toString();
    String evntDesc = "Dummy Automation Test";
    String evntS = "evntS";
    String d_type = "type";
    String b_area = "area";
```

```
String pfx = "pfx";
String obj_s = "s";
String mimeType = "auto";
String sys_id = "Auto test";
String doc = "auto";
String GUID = '{' + generateUUID.toString() + '}';

@Test
public void producerMessageSimpleAgrtKeyHappyTest() throws
    ExecutionException, InterruptedException {
    KafkaProducerSetup kafkaProducerSetup = new KafkaProducerSetup();

    logger.info("Adding properties to Kafka Producer...");

    Properties props = kafkaProducerSetup.setProducerProps(
    ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.BOOTSTREP_SERVER_QA)),
        ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.AUTH_INFO_SYSTEM)),
        ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.SASL_CONFIG_SYSTEM)),
        ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.TRUST_STORE_LOCATION)),
        ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.SCHEMA_REGISTRY)));

    GenericRecord message = KafkaProducerSetup.produceMessage(eventType,
        eventSubtype, currentDateComplete, evntId, evntDesc, evntS,
        d_type, b_area, pfx, obj_s,
    mimeType, sys_id, doc, GUID);

    logger.info("Preparing/Sending the object to Kafka Producer");

    ProducerRecord<Object, Object> record =
    new ProducerRecord<>(ConfigUtils.
    getProperty(String. valueOf(ConfigUtils.ConfigKeys.
    TOPIC_NAME_PRODUCER)), null, message);
```

```
        KafkaProducer<Object, Object> producer = new KafkaProducer<>(props);
        RecordMetadata metadata = producer.send(record).get();
        producer.flush();
        producer.close();

        logger.info("Message sent successfully to topic" +
        metadata.topic() + " " + "partition: " + metadata.partition()
         + " " + "offset: " + metadata.offset());
    }
}
```

Test Results Kafka Producer Overview:

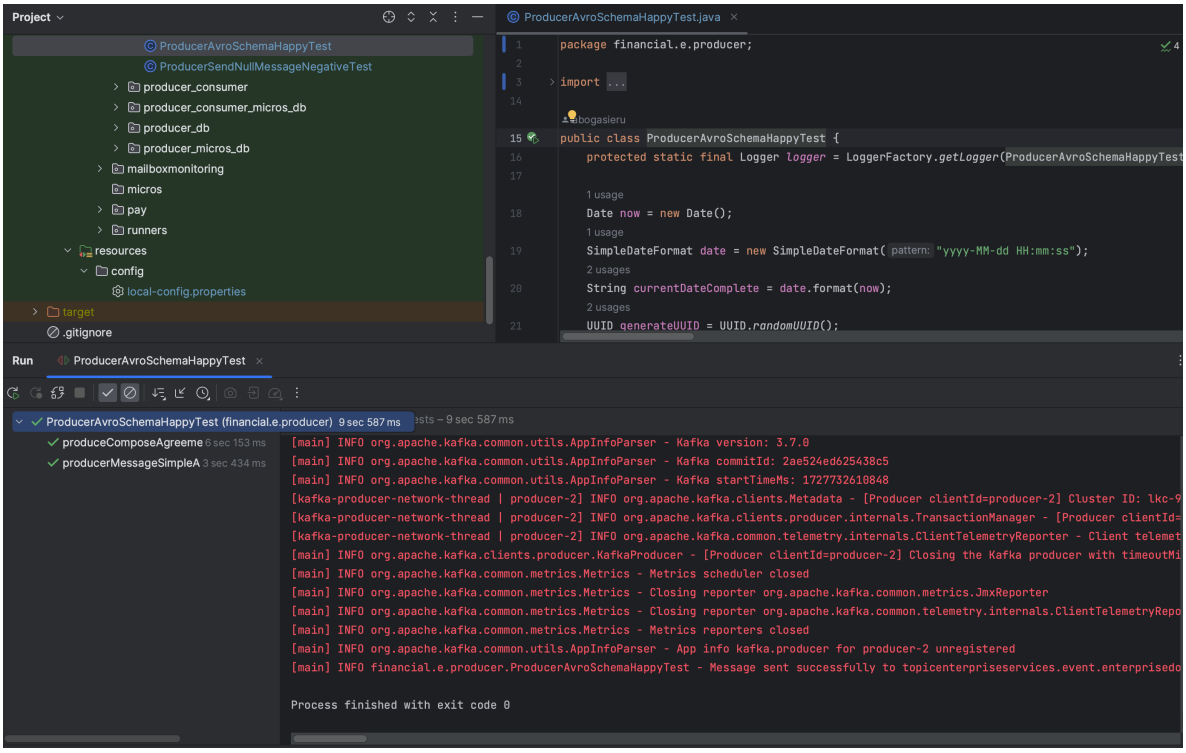

Figure 6.5: Kafka Producer Overview

With Kafka Producer and Kafka Consumer functional tests we can implement end-to-end test between Kafka Producer and Kafka Consumer by generating message against Avro schema and validate it on the Kafka Consumer side.

1.c) Kafka Producer - Kafka Consumer integration test steps:

- Set properties for Kafka Producer;
- Generate a valid Avro schema message;
- Message is successfully published on topic Producer: partition and offset;
- Set properties for Kafka Consumer;

- Subscribe to the Consumer topic;
- Consume and validate published message fields by evntId field.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.
producer_consumer;

import financial.kafka.KafkaConsumerSetup;
import financial.kafka.KafkaProducerSetup;
import fiancial.utils.general.ConfigUtils;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;
import java.util.Properties;
import java.util.UUID;
import java.util.concurrent.ExecutionException;

import static org.assertj.core.api.Assertions.assertThat;

public class ProducerAvroSchemaConsumerValidationsHappyTest
{
    protected static final Logger logger = LoggerFactory.getLogger(
    ProducerAvroSchemaConsumerValidationsHappyTest.class);
    Date now = new Date();
    SimpleDateFormat date = new SimpleDateFormat(
    "yyyy-MM-dd HH:mm:ss");
    String currentDateComplete = date.format(now);
```

```
UUID generateUUID = UUID.randomUUID();
String eventType = "ev";
String eventSubtype = "stype";
String evntId = generateUUID.toString();
String evntDesc = "Auto";
String evnt_s = "evnt";
String d_type = "type";
String b_area = "area";
String pfx = "pfx";
String obj_s = "autoS";
String m_type = "auto";
String sys_id = "auto";
String tran_id = "auto";
String prim_id = "prim";
String adm_sys = "sys";
String doc = "aut";
String GUID = '{' + generateUUID.toString() + '}';
String pattern = "mm-dd-yyyy";
SimpleDateFormat dateFormat = new SimpleDateFormat(pattern);
String currentDate = dateFormat.format(new Date());
String UNIQUE_GROUP_ID = "automation_events_consumer"
+ currentDate;



@Test
public void producerMessageConsumerValidationsHappyTest() throws
    ExecutionException, InterruptedException {
    KafkaProducerSetup kafkaProducerSetup = new KafkaProducerSetup();

    logger.info("Adding properties to Kafka Producer...");

    Properties propsProducer = kafkaProducerSetup.setProducerProps(
    ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.BOOTSTREP_SERVER_QA)),
        ConfigUtils.getProperty(String.valueOf(
```

```
ConfigUtils.ConfigKeys.AUTH_INFO_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.SASL_CONFIG_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.TRUST_STORE_LOCATION)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.SCHEMA_REGISTRY)));
GenericRecord message = KafkaProducerSetup.produceMessage(eventType,
eventSubtype, currentDateComplete, evntId,
evntDesc, evnt_s, d_type, b_area, pfx, obj_s, m_type, sys_id, doc,
    GUID);

logger.info("Preparing/Sending the object to Kafka Producer");

ProducerRecord<Object, Object> recordProducer =
new ProducerRecord<>(ConfigUtils.getProperty(
String.valueOf(ConfigUtils.ConfigKeys.
TOPIC_NAME_PRODUCER)), null, message);

KafkaProducer<Object, Object> producer = new
    KafkaProducer<>(propsProducer);
RecordMetadata metadata = producer.send(recordProducer).get();
producer.flush();
producer.close();

logger.info("Message sent successfully to topic" + metadata.topic() +
    " " + "partition: " + metadata.partition() + " " + "offset: " +
    metadata.offset());

KafkaConsumerSetup kafkaConsumerSetup = new KafkaConsumerSetup();

logger.info("Adding properties to Kafka Consumer...");
Properties propsConsumer =
    kafkaConsumerSetup.setConsumerProps(ConfigUtils.
getProperty(String.valueOf(
ConfigUtils.ConfigKeys.BOOTSTREP_SERVER_QA)),
UNIQUE_GROUP_ID, ConfigUtils.getProperty(String.valueOf(
```

```
ConfigUtils.ConfigKeys.AUTH_INFO_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.SASL_CONFIG_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.TRUST_STORE_LOCATION)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.SCHEMA_REGISTRY)));

logger.info("Initializing and subscribing to the topic");
KafkaConsumer<String, GenericRecord> consumer =
new KafkaConsumer<>(propsConsumer);
consumer.subscribe(Arrays.asList(ConfigUtils.
getProperty(String.valueOf(ConfigUtils.
ConfigKeys.TOPIC_SYSTEM_CONSUMER))));

ConsumerRecords<String, GenericRecord> records = consumer.poll(1000);

for (ConsumerRecord<String, GenericRecord> record : records) {
GenericRecord avroRecordConsum = record.value();

logger.info("Current record value: " + record.value().toString());
String evntIdConsumer = avroRecordConsum.
get("eventHeader.evntId").toString();

if(evntId.equals(evntIdConsumer)) {
assertThat(avroRecordConsum.
.toString()).
isEqualTo(eventType);

assertThat(avroRecordConsum.
get("eventHeader.eventSubtype").
toString()).isEqualTo(eventSubtype);

assertThat(avroRecordConsum.
get("eventHeader.eventDateTime").
toString()).isEqualTo(currentDateComplete);
assertThat(avroRecordConsum.
```

```
get("eventHeader.eventGeneratedDateTime")
.toString()).isEqualTo(currentDateComplete);

assertThat(avroRecordConsum.
get("eventHeader.evntId").
toString()).isEqualTo(evntId);

assertThat(avroRecordConsum.
get("eventHeader.eventRequestId") .toString()).isEqualTo(evntId);

assertThat(avroRecordConsum.
get("eventHeader.evntDesc")
.toString()).isEqualTo(evntDesc);

assertThat(avroRecordConsum.
get("eventBody.b_area")
.toString()). isEqualTo(b_area);

assertThat(avroRecordConsum.
get("eventBody.obj_s").
toString()).isEqualTo(obj_s);

assertThat(avroRecordConsum.
get("eventBody.m_type").
toString()).isEqualTo(m_type);

assertThat(avroRecordConsum.
get("eventBody.sys_id").toString()).
isEqualTo(sys_id);

//remaining assertions
}
long offset = record.offset();
logger.info("Consumed record at offset: " + offset);
}
consumer.close();
}
```

```
}
```

Kafka Producer - Consumer test results:

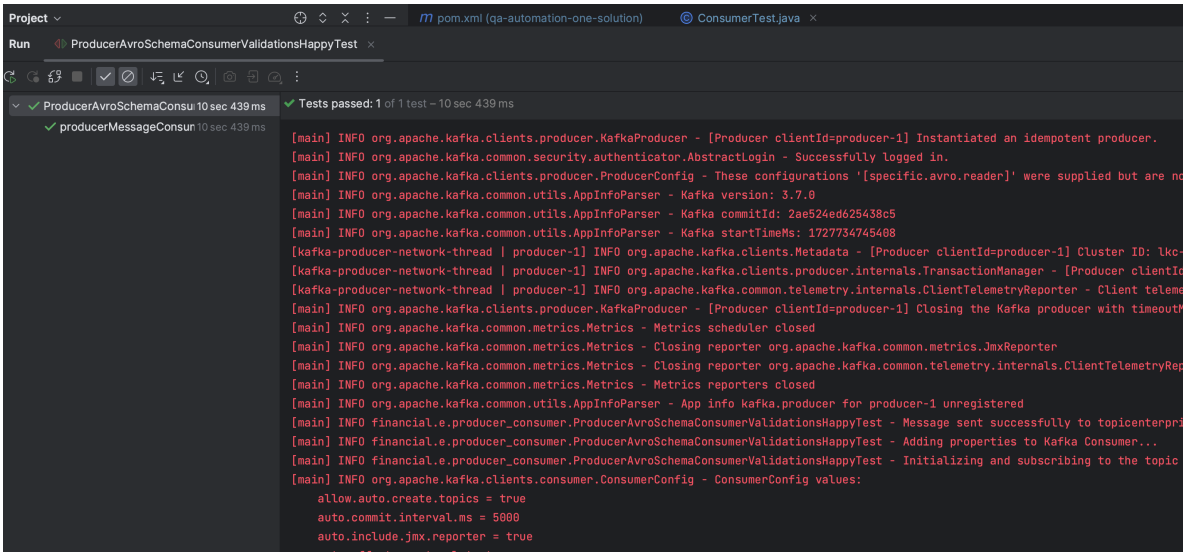

Figure 6.6: Kafka Producer - Kafka Consumer test results

2.a) GET Agr Cst test **RESTful API** steps:

- Make a call to Agr Cst RESTful API;
- Parse the response message;
- Validate and save roleType and mbrGUIDAgrCst values fields.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.micros;

import io.restassured.response.Response;
import financial.utils.connector.RestAssuredConnector;
import financial.utils.general.ConfigUtils;
import org.json.JSONArray;
import org.json.JSONObject;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import java.util.Map;
import static org.junit.Assert.assertEquals;

public class GETAgrCstHappyTest {
    protected static final Logger logger =
        LoggerFactory.getLogger(GETAgrCstHappyTest.class);
    private String requestUri;
    private String END_URL_AGR_QA="99999";

    @Test
    public void GETAgrCstHappyTest() {
        requestUri = ConfigUtils.getProperty(ConfigUtils.
        ConfigKeys.BASE_URL_QA.toString())
        +
       ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
        MIDDLE_URL_AGR_QA.toString()) + END_URL_AGR_QA;
        logger.info("requestUri:" + " " + requestUri);

        RestAssuredConnector connector = new RestAssuredConnector();
        Map<String, String> headers = RestAssuredConnector.setHeaders(
        ConfigUtils.getProperty(
        ConfigUtils.ConfigKeys.TOKEN_AG_CSTR.toString()));
        Response agrRsp = connector.getRequest(requestUri, headers);

        String rsp_agrKey = agrRsp.getBody().asString();
        logger.info("Response:" + " " + rsp_agrKey);

        assertEquals(200, agrRsp.getStatusCode());
        JSONObject jsonResponse = new JSONObject(rsp_agrKey);

        JSONArray agrts = jsonResponse.getJSONArray("agrts");
        JSONObject cst = agrts.getJSONObject(0).getJSONArray("agrCst").
        getJSONObject(0);

        JSONArray agrtsArray = jsonResponse.getJSONArray("agrts");
        String agrtKeyMS = agrtsArray.getJSONObject(0).getString("agrtKey");
```

```
        String roleType = cst.getString("roleType");
        String mbrGUIDAgrCst = cst.getString("mbrGUID");

        logger.info("Fields extracted from the ms:" + "agrkey: " + agrtKeyMS +
            " " + "roleType: " + roleType + " " + "mbr GUID:" + mbrGUIDAgrCst);
    }
}
```

2.b) **Cst Id** test RESTful API steps:

- Make call to Cst Id RESTful API;
- Parse response message;
- Validate and save mbrGUID value field.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.micros;

import io.restassured.response.Response;
import massmutual.utils.connector.RestAssuredConnector;
import massmutual.utils.general.ConfigUtils;
import org.json.JSONArray;
import org.json.JSONObject;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.Map;
import static org.junit.Assert.assertEquals;

public class GETCstIdHappyTest {
    protected static final Logger logger =
        LoggerFactory.getLogger(GETCstIdHappyTest.class);
    private String requestUri;
    private String CST_ID = "DEMO";

    @Test
    public void getCustomerIdHappyTest() {
```

```
        requestUri = ConfigUtils.getProperty(ConfigUtils.
        ConfigKeys.BASE_URL_QA.toString())
        +
        ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
        MIDDLE_URL_CST_QA.toString()) + CUSTOMER_ID;

        RestAssuredConnector connector = new RestAssuredConnector();

        Map<String, String> headers = RestAssuredConnector.setHeaders(
        ConfigUtils.getProperty(
        ConfigUtils.ConfigKeys.TOKEN_CST_ID.toString()));

        Response cstrIdResponse = connector.getRequest(requestUri, headers);
        logger.info("Cst Url:" + " " + requestUri);

        String responseBody_cstId = cstIdResponse.getBody().asString();
        logger.info("Response:" + " " + responseBody_cstrId);

        assertEquals(200, cstIdResponse.getStatusCode());
        logger.info("Accessing mbrGUID..");

        JSONObject jsonResponse = new JSONObject(responseBody_cstId);

        JSONArray cst = jsonResponse.getJSONArray("cst");
        String mbrGUID = cst.getJSONObject(0).getString("mbrGUID");

        logger.info("mbrGUID: " + mbrGUID);
    }
}
```

3. **Kafka Producer - DB: integration test** to cover unhappy path: no record found in the database having an invalid type.
Steps:

- Set properties on Kafka Producer;
- Generate message having invalid dType value field;

- Connect to the database;
- No record is found for our type: "auto" for the message produced.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.
producer_db;

import financial.db.DBConnections;
import financial.kafka.KafkaProducerSetup;
import financial.utils.general.ConfigUtils;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.json.JSONObject;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Properties;
import java.util.UUID;
import java.util.concurrent.ExecutionException;

public class ProducerAvroSchemaDBValidationsType
AutoAndroidUnhappyTest {
protected static final Logger logger = LoggerFactory.getLogger(
    ProducerAvroSchemaDBValidationsTypeAutoAndroid.class);

    Date now = new Date();
    SimpleDateFormat date = new SimpleDateFormat(
    "yyyy-MM-dd HH:mm:ss");
    String currentDateComplete = date.format(now);
    UUID generateUUID = UUID.randomUUID();
```

```
String eventType = "enterpriseservices";
String eventSubtype = "documentaddupdate";
String evntId = generateUUID.toString();
String evntDesc = "Auto";
String evntS = "Auto";
String dType = "DUMMY";
String bArea = "Auto";
String pfx = "pfx";
String objS = "AUTO";
String mimeType = "app";
String sysId = "Auto";
String primId = "999";
String doc = "pfx";
String GUID = '{' + generateUUID.toString() + '}';
String appIdExp = "AUTO";
String msgType = "auto";
DBConnections dbConn;
String QUERY = "SELECT ident, type, format, source, received_date,
    status, payload FROM message WHERE type
= \"auto\" AND source = \"AUTO\" AND JSON_EXTRACT(payload,
    \"$.content.data.appIdType\")=\"androidAppId\" AND
    JSON_EXTRACT(payload, \"$.content.data.origEvId\")=?";

@Test
public void produceMessageDBValidationsTypeAutoAndroid
UnHappyTest() throws ExecutionException, InterruptedException,
    SQLException {
    KafkaProducerSetup kafkaProducerSetup = new KafkaProducerSetup();

    logger.info("Adding properties to Kafka Producer...");

    Properties props = kafkaProducerSetup.setProducerProps(ConfigUtils.
    getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.BOOTSTREP_SERVER_QA)),
        ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.AUTH_INFO_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
```

```
ConfigUtils.ConfigKeys.SASL_CONFIG_SYSTEM)),
ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.TRUST_STORE_LOCATION)),
ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.SCHEMA_REGISTRY)));

GenericRecord message = KafkaProducerSetup.
produceMessageComposeAgreementKey(eventType, eventSubtype,
    currentDateComplete, evntId, evntDesc, evntS, dType, bArea, pfx,
    objS, mimeType, sysId, doc, GUID);

logger.info("Preparing/Sending the object to Kafka Producer");

ProducerRecord<Object, Object> record = new ProducerRecord<>
    (ConfigUtils.getProperty( String.valueOf(ConfigUtils.ConfigKeys.
TOPIC_NAME_PRODUCER)), null, message);

KafkaProducer<Object, Object> producer = new KafkaProducer<>(props);
RecordMetadata metadata = producer.send(record).get();
producer.flush();
producer.close();

logger.info("Message sent successfully to topic" + metadata.topic() +
    " " + "partition: " + metadata.partition() + " " + "offset: " +
metadata.offset());

logger.info("Validating the DB payload:");
dbConn = new DBConnections();
ResultSet rs = dbConn.dbConn(ConfigUtils.
getProperty(String.valueOf(
ConfigUtils.ConfigKeys.DB_URL_QA)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.DB_USERNAME_QA)),
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.DB_PASSWORD_QA)),
QUERY, evntId);
```

```
        if (!rs.next()) {
            logger.info("There is no message in DB");
        } else {
            logger.info("There is something wrong and there is a message in
                DB");
        }
        dbConn.closeConnection(rs.getStatement().
        getConnection());
    }
}
```

---

4. Kafka Producer - RESTful APIs - DB integration test steps:

- Set properties for Kafka Producer;
- Generate a valid Avro schema message;
- Message is successfully published on the partition and offset topic Producer;
- Make a call to Agr Cst RESTful API;
- Parse the response message;
- Validate and save roleType and mbrGUIDAgrCst values fields;
- Make call to Cst Id RESTful API;
- Parse response message;
- Validate and save mbrGUID value field;
- Connect to the database;
- Database result with type: "auto" for the message produced is found;
- Validations for the DB columns against message produced fields.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.producer_
micros_db;

import io.restassured.response.Response;
import financial.db.DBConnections;
```

```
import financial.kafka.KafkaProducerSetup;
import financial.utils.connector.RestAssuredConnector;
import financial.utils.general.ConfigUtils;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.json.JSONArray;
import org.json.JSONObject;
import org.junit.Before;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.Properties;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import static org.junit.Assert.assertEquals;

public class ProducerAvroSchemaMsDBValidationsTypeAutoHappyTest {
protected static final Logger logger = LoggerFactory.getLogger(
    ProducerAvroSchemaMsDBValidationsTypeAutoHappyTest.
class);

    Date now = new Date();
    SimpleDateFormat date = new SimpleDateFormat(
    "yyyy-MM-dd HH:mm:ss");
    String currentDateComplete = date.format(now);
    UUID generateUUID = UUID.randomUUID();
    String eventType = "auto";
    String eventSubtype = "doc";
    String evntId = generateUUID.toString();
    String evntDsc = "Auto";
```

```
String evnt_s = "auto";
String doc_type = "auto";
String b_area = "auto";
String pfx = "pfx";
String obj_s = "auto";
String m_type = "app";
String sys_is = "auto";
String doc = "auto";
String GUID = '{' + generateUUID.toString() + '}';
String appIdExp = "auto";
String msg_t = "auto";
DBConnections dbConn;
private String requestUriAgrCstr;
private String requestUriCstrId;
String exptRole = "auto";
public RestAssuredConnector connector;
public Response agrRsp;
public Response cstrIdResponse;
public String mbrGUIDcstrId;
public String QUERY = "SELECT * FROM message WHERE type = \"auto\" AND
    source = \"auto\" AND JSON_EXTRACT(payload, \"$.triggers.onsuccess\")
    = \"auto\" AND
JSON_EXTRACT(payload, \"$.content.data.origEvntId\")= ?";
private final String agr_key="999";
private final String agr_key_end="auto";
private String CST_ID = "auto";

@Before
public void setupData() {
dbConn = new DBConnections();
requestUriCstrId = ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
BASE_URL_QA.toString())
+
ConfigUtils.getProperty(ConfigUtils.
ConfigKeys.MIDDLE_URL_CSTRID_QA.toString()) + CST_ID;

requestUriAgrCstr = ConfigUtils.getProperty(ConfigUtils.ConfigKeys.
```

```
    BASE_URL_KONG_QA.toString()) +
    ConfigUtils.getProperty(ConfigUtils.
    ConfigKeys.MIDDLE_URL_AGR_QA.toString()) + agr_key
    + "1111" + agr_key_end;

    connector = new RestAssuredConnector();
}
    @Test
    public void sendKafkaMessageDbValidationsHappyTest() throws
        ExecutionException, InterruptedException, SQLException {
    KafkaProducerSetup kafkaProducerSetup = new KafkaProducerSetup();

    logger.info("Adding properties to Kafka Producer...");

    Properties props = kafkaProducerSetup.setProducerProps(ConfigUtils.
    getProperty(String.valueOf(ConfigUtils.
    ConfigKeys.BOOTSTREP_SERVER_QA)),
    ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.AUTH_INFO_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.SASL_CONFIG_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.TRUST_STORE_LOCATION)),
    ConfigUtils.getProperty(String.valueOf(
    ConfigUtils.ConfigKeys.SCHEMA_REGISTRY)));

    GenericRecord message = KafkaProducerSetup.produceMessage(eventType,
    eventSubtype, currentDateComplete, evntId,
    evntDsc, evnt_s, doc_type, b_area, pfx, obj_s, m_type, sys_is, doc, GUID);

    logger.info("Preparing/Sending the object to Kafka Producer");
    ProducerRecord<Object, Object> record = new ProducerRecord<>(ConfigUtils.
    getProperty(String.valueOf(ConfigUtils.ConfigKeys
    TOPIC_NAME_PRODUCER)), null, message);

    KafkaProducer<Object, Object> producer = new KafkaProducer<>(props);
    RecordMetadata metadata = producer.send(record).get();
```

```
producer.flush();
producer.close();

logger.info("Message sent successfully to topic" + metadata.topic() + " "
    +
"partition: " + metadata.partition() + " " + "offset: " +
    metadata.offset());

logger.info("Agr cstr call:");
Map<String, String> headers = RestAssuredConnector.setHeaders(ConfigUtils.
getProperty(ConfigUtils.ConfigKeys.
TOKEN_AGR_CSTR.toString()));

Response agrRsp = connector.getRequest(requestUriAgrCstr, headers);

logger.info("Extracting cstr, agr key, role type
and mbr GUID");
String rspBody_agrKey = agrRsp.getBody().asString();

JSONObject jsonResponse = new JSONObject(
rspBody_agrKey);

JSONArray agrts = jsonResponse.
getJSONArray("agrts");
JSONObject cstr = agrts.getJSONObject(0).getJSONArray("agrCstr").
getJSONObject(0);

JSONArray agrtsArray = jsonResponse.getJSONArray("agrts");
String agrKeyMS = agrtsArray.getJSONObject(0).getString("agrKey");

String roleType = cstr.getString("roleType");

Map<String, String> headerscstr =
    RestAssuredConnector.setHeaders(ConfigUtils.
getProperty(ConfigUtils.ConfigKeys.

Response cstrIdResponse = connector.
```

```
getRequest(requestUriCstrId, headerscstr);

String responseBody_cstrId = cstrIdResponse.getBody().asString();
JSONObject cstrResponse = new JSONObject(responseBody_cstrId);

JSONArray cstrs = cstrResponse.getJSONArray("cstrs");

String mbrGUIDcstrId = cstrs.getJSONObject(0).getString("mbrGUID");

assertThat(roleType).isEqualTo(exptRole);
assertThat(agrKeyMS).isEqualTo(agr_key + agr_key_end);

logger.info("Validating the DB payload:");

ResultSet rs = dbConn.dbConn(ConfigUtils.
getProperty(
ConfigUtils.ConfigKeys.DB_URL_QA.name()),
ConfigUtils.getProperty(
String.valueOf(ConfigUtils.ConfigKeys
.DB_USERNAME_QA)),
ConfigUtils.getProperty(String.valueOf(
ConfigUtils.ConfigKeys.DB_PASSWORD_QA)), QUERY, evntId);

if (rs.next()) {
    String type = rs.getString("type");
    String source = rs.getString("source");
    String payload = rs.getString("payload");
    String received_date = rs.getString("received_date");

    logger.info("Validating the DB payload:");

    JSONObject jsonPayload = new JSONObject(payload);
    JSONObject data = jsonPayload.getJSONObject("content").
    getJSONObject("data");
    String origEvntId = data.getString("origVntId");
    String appId = jsonPayload.getString("appId");
    String overwritemsg_t = data.getString("overwritemsg_t");
```

```
        //extract all the fields

        JSONArray onsuccess = jsonPayload.getJSONObject("triggers").
        getJSONArray("onsuccess");
        String Dbmsg_t = jsonPayload.getString("msg_t");

        if (origEvntId.equals(evntId)) {

            assertThat(origAppId).isEqualTo(appIdExp);

            assertThat(overwritemsg_t).isEqualTo(msg_t);
            assertThat(origEvntInitDesc).isEqualTo(evntDsc);

            assertThat(originalevntDsc).isEqualTo(evntDsc);

           assertThat(Dbmsg_t).isEqualTo(msg_t);
       //do all the remaining assertions
            } else {
            logger.info("Something must be really wrong with this data");
            }
        }
        dbConn.closeConnection(rs.getStatement().
        getConnection());
    }
}
```

---

5. Producer - Consumer - RESTful APIs - DB end-to-end test steps:

- Set properties for Kafka Producer;
- Generate a valid Avro schema message;
- Message is successfully published on the partition and offset topic Producer;
- Set properties for Kafka Consumer;
- Subscribe to the Consumer topic;
- Consume and validate published message fields by evntId field;

- Make a call to Agr Cst RESTful API;
- Parse the response message;
- Validate and save roleType and mbrGUIDAgrCst values fields;
- Make call to Cst Id RESTful API;
- Parse response message;
- Validate and save mbrGUID value field;
- Connect to the database;
- Database result with type: "auto" for the message produced is found;
- Validations for the DB columns against message produced fields.

```
package financial.KafkaProducerConsumerRESTfulAPIsDB.producer_
consumer_micros_db;

import io.restassured.response.Response;
import financial.db.DBConnections;
import financial.kafka.KafkaConsumerSetup;
import financial.kafka.KafkaProducerSetup;
import financial.utils.connector.RestAssuredConnector;
import financial.utils.general.ConfigUtils;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.json.JSONArray;
import org.json.JSONObject;
import org.junit.Before;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.ExecutionException;

import static org.assertj.core.api.Assertions.assertThat;

public class ProducerConsumerAgrMsDBValidationsHappyTest {

    protected static final Logger logger = LoggerFactory.getLogger(
    ProducerConsumerAgrMsDBValidationsHappyTest.class);

    Date now = new Date();
    SimpleDateFormat date = new SimpleDateFormat(
    "yyyy-MM-dd HH:mm:ss");
    String currentDateComplete = date.format(now);
    UUID generateUUID = UUID.randomUUID();
    String eventType = "e";
    String eventSubtype = "ev";
    String evId = generateUUID.toString();
    String ev_desc = "Auto";
    String evntS = "auto";
    String d_type = "auto";
    String b_area = "auto";
    String pfx = "pfx";
    String o_st = "auto";
    String m_type = "app";
    String sys_id = "auto";
    String tran_id = "auto";
    String prim_id = "auto";
    String adm_sys = "auto";
    String doc = "auto";
    String GUID = '{' + generateUUID.toString() + '}';
    String pattern = "mm-dd-yyyy";
    SimpleDateFormat dateFormat = new SimpleDateFormat(pattern);
    String currentDate = dateFormat.format(new Date());
```

```
String UNIQUE_GROUP_ID = "automation_events_consumer" + currentDate;
public RestAssuredConnector connector;
DBConnections dbConn;
private String requestUriAgrCst;
private final String agr_key = "99999";
private final String agr_key_end = "AUTO";

public String QUERY = "SELECT * FROM message WHERE type = \"auto\" AND
    source = \"source\" AND JSON_EXTRACT(payload, \"$.triggers.onsucces\")
    = \"auto\" AND s
JSON_EXTRACT(payload, \"$.content.data.origId\")= ?";

@Before
public void setup() {
    requestUriAgrCst = ConfigUtils.getProperty(
    ConfigUtils.ConfigKeys.BASE_URL_QA. toString())
    +
   ConfigUtils.getProperty(ConfigUtils.
   ConfigKeys.MIDDLE_URL_AGR_QA.toString())
    +
    agr_key + "-" + agr_key_end;
    connector = new RestAssuredConnector();
    dbConn = new DBConnections();
}

@Test
public void produceMessageConsumerAgrMsDBValidationsTest()
throws ExecutionException, InterruptedException, SQLException {

KafkaProducerSetup kafkaProducerSetup = new KafkaProducerSetup();

logger.info("Adding properties to Kafka Producer...");

Properties propsProducer =
    kafkaProducerSetup.setProducerProps(ConfigUtils.
getProperty(String.valueOf(ConfigUtils.
ConfigKeys.BOOTSTREP_SERVER_QA)),
```

```
ConfigUtils.getProperty(String.valueOf(ConfigUtils.
ConfigUtils.getProperty(String.valueOf(ConfigUtils.
ConfigKeys.SASL_CONFIG_SYSTEM)),
    ConfigUtils.getProperty(String.valueOf(ConfigUtils.
ConfigKeys.TRUST_STORE_LOCATION)),
    ConfigUtils.getProperty(String.valueOf(ConfigUtils.
ConfigKeys.SCHEMA_REGISTRY)));

GenericRecord message = KafkaProducerSetup.
produceMessage(eventType, eventSubtype, currentDateComplete,
evId, ev_desc, evntS, d_type, b_area, pfx, o_st, m_type, sys_id, doc,
    GUID);

logger.info("Preparing/Sending the object to Kafka Producer");

ProducerRecord<Object, Object> recordProducer =
new ProducerRecord<>(ConfigUtils.getProperty(
String.valueOf(ConfigUtils.
ConfigKeys.TOPIC_NAME_PRODUCER)), null, message);

KafkaProducer<Object, Object> producer = new
    KafkaProducer<>(propsProducer);

RecordMetadata metadata = producer.send(recordProducer).get();
producer.flush();
producer.close();

logger.info("Message sent successfully to topic" + metadata.topic() +
" " + "partition: " + metadata.partition() + " " + "offset: " +
metadata.offset());

KafkaConsumerSetup kafkaConsumerSetup = new KafkaConsumerSetup();

logger.info("Adding properties to Kafka Consumer...");
Properties propsConsumer =
    kafkaConsumerSetup.setConsumerProps(ConfigUtils.
getProperty(String.valueOf(ConfigUtils.ConfigKeys.
```

```
BOOTSTREP_SERVER_QA)), UNIQUE_GROUP_ID,
    ConfigUtils.getProperty(String.valueOf(
ConfigUtils.
ConfigKeys.AUTH_INFO_SYSTEM)), ConfigUtils.getProperty(String.
valueOf(ConfigUtils.ConfigKeys
.SASL_CONFIG_SYSTEM)),
ConfigUtils.getProperty(String.valueOf(ConfigUtils.
ConfigKeys.TRUST_STORE_LOCATION)),
    ConfigUtils.getProperty(String.valueOf(ConfigUtils.
ConfigKeys.SCHEMA_REGISTRY)));

logger.info("Initializing and subscribing to the topic");

KafkaConsumer<String, GenericRecord> consumer =
new KafkaConsumer<>(propsConsumer);

consumer.subscribe(Arrays.asList(ConfigUtils.
getProperty(String.valueOf(ConfigUtils.ConfigKeys.
    TOPIC_SYSTEM_CONSUMER))));

ConsumerRecords<String, GenericRecord> records = consumer.poll(1000);
for (ConsumerRecord<String, GenericRecord> record : records) {
GenericRecord avroRecordConsum = record.value();

logger.info("Current record value: " + record.value().toString());

String evIdConsumer = avroRecordConsum.get("eventHeader.evId").
toString();

if (evId.equals(evIdConsumer)) {
assertThat(avroRecordConsum.
get("eventHeader.eventType").
toString()).isEqualTo(eventType);
assertThat(avroRecordConsum.get("eventHeader.
eventSubtype").toString()).isEqualTo(eventSubtype);
assertThat(avroRecordConsum.get("eventHeader.
eventDateTime").toString()).
```

```
assertThat(avroRecordConsum.get("eventHeader.
eventGeneratedDateTime").toString()).
assertThat(avroRecordConsum.get(
"eventHeader.evId").toString()).isEqualTo(evId);
assertThat(avroRecordConsum.get(
"eventHeader.ev_desc").
toString()).isEqualTo(ev_desc);
assertThat(avroRecordConsum.get("eventBody.b_area").
    toString()).isEqualTo(b_area);
assertThat(avroRecordConsum.get("eventBody.o_st").
    toString()).isEqualTo(o_st);
assertThat(avroRecordConsum.get("eventBody.m_type").
    toString()).isEqualTo(m_type);
assertThat(avroRecordConsum.get("eventBody.sys_id").
    toString()).isEqualTo(sys_id);
assertThat(avroRecordConsum.get("eventBody.adm_sys")
    .toString()).isEqualTo(adm_sys);
   }
long offset = record.offset();
logger.info("Consumed record at offset: " + offset);
  }
consumer.close();

logger.info("Validating the Consumer fields against the fields
    produced... with success, messages is sent on Consumer side");

logger.info("Agr cstr call:");

Map<String, String> headers = RestAssuredConnector.setHeaders(ConfigUtils.
getProperty(ConfigUtils.ConfigKeys.
TOKEN_AGR_CSTR.toString()));

Response agrResponse = connector.getRequest(requestUriAgrCst, headers);

logger.info("Extracting cstr, agr key, role type
and mbr GUID");
```

```
String responseBody_agrtKey = agrResponse.getBody().asString();
JSONObject jsonResponse = new JSONObject(responseBody_agrKey);

JSONArray agr = jsonResponse.getJSONArray("agr");
JSONObject cstr = agr.getJSONObject(0).getJSONArray("agrcstrs").
getJSONObject(0);

JSONArray agrArray = jsonResponse.getJSONArray("agr");
String agrKeyMS = agrArray.getJSONObject(0).getString("agrKey");

String roleType = cstr.getString("roleType");
String mbrGUIDAgrCstr = cstr.getString("mbrGUID");
logger.info(mbrGUIDAgrCstr);

assertThat(roleType).isEqualTo("OWNR");
assertThat(agrKeyMS). isEqualTo(agr_key + agr_key_end);
ResultSet rs = dbConn.dbConn(ConfigUtils.getProperty(String.
valueOf(ConfigUtils.ConfigKeys.DB_URL_QA)), ConfigUtils.getProperty(
String.valueOf(ConfigUtils.
ConfigKeys.DB_USERNAME_QA)),ConfigUtils.getProperty(

String.valueOf(ConfigUtils.
ConfigKeys.DB_PASSWORD_QA)), QUERY, evId);

if (rs.next()) {
    String type = rs.getString("type");
    String source = rs.getString("source");
    String payload = rs.getString("payload");
    String received_date = rs.getString("received_date");

    logger.info("Validating the DB payload:");

    JSONObject jsonPayload = new JSONObject(payload);
    JSONObject data = jsonPayload.getJSONObject("content").
    getJSONObject("data");

    //extract all the fields
```

```
        String origvId = data.getString("origId");
        String appId = jsonPayload.getString("appId");

        JSONArray onsuccess = jsonPayload.getJSONObject("triggers").
        getJSONArray("onsuccess");
        String DbmsgType = jsonPayload.getString("msgType");

        if (origId.equals(evId)) {

            assertThat(origAppId).isEqualTo("AUTO");
            assertThat(msgType).isEqualTo("auto");
            assertThat(origEventInitDesc).
            isEqualTo(ev_desc);

            assertThat(originalev_desc).
            isEqualTo(ev_desc);

            assertThat(origEventSrc).isEqualTo("AUTO NOW");

            assertThat(originalEventDateTime).
            isEqualTo(currentDateComplete);

            assertThat(originalEventInitiator).
            isEqualTo(applicationId);
            assertThat(DbmsgType).isEqualTo("auto");
            } else {
                logger.info("Something must be really wrong with this data");
            }
        }
        dbConn.closeConnection(rs.getStatement().
        getConnection());
    }
}
```

Case2: RESTful APIs and PostgreSQL database systems.

1. Data modeling for RESTful API, as we are already familiars with this type of approach we are creating a JSON Object from payload mapping each field with its value to make the a post request.

```
package financial.restfulapis;

import io.restassured.response.Response;
import org.json.JSONArray;
import org.json.JSONObject;

public class RESTfulAPIDataModel {
    public String createModel(String email, String type, String guid, int no,
        long amount, int r_no, String rec_type, String tr_type, String key,
        String method, String name){

        JSONObject restfulPayload = new JSONObject();
        restfulPayload.put("transGuid", guid);
        restfulPayload.put("code", 1);
        restfulPayload.put("email", email);
        restfulPayload.put("type", type);
        restfulPayload.put("cId", guid);
        restfulPayload.put("no", no);
        restfulPayload.put("amount", amount);
        restfulPayload.put("rNo", r_no);
        restfulPayload.put("trId", guid);
        restfulPayload.put("recType", rec_type);
        restfulPayload.put("type", type);
        restfulPayload.put("key", key);
        restfulPayload.put("method", method);
        restfulPayload.put("auth_type", "auto");
        restfulPayload.put("sys", "auto");
        restfulPayload.put("date", "2024-02-13 01:06:84532 PM");
        restfulPayload.put("id", "auto");
        restfulPayload.put("name", name);
        return restfulPayload.toString();
    }
```

```
    public int saveKey(Response responseBody_restful){
        JSONObject payloadObject = new
            JSONObject(responseBody_restful.asString());
        JSONArray msgArray = jsonObject.getJSONArray("messages");
        JSONObject msgObject = msgArray.getJSONObject(0);
        String msgDesc = msgObject.getString("msgDesc");
        String keyString = msgDesc.substring(msgDesc.lastIndexOf(" " ) + 1);
        int keyInt = Integer.parseInt(keyString.trim());
        return keyString;
    }
}
```

---

1. RESTful API test steps:

- Make call to RESTful API having the payload generated in data modeling from **RESTfulAPIDataModel** class;
- Parse and validate response message.

RESTful API test implementation:

```
package financial.RESTfulAPIsDB.micros;

import io.restassured.response.Response;
import financial.db.DBConnections;
import financial.restfulapis.RESTfulAPIDataModel;
import financial.utils.connector.RestAssuredConnector;
import financial.utils.general.ConfigUtils;
import org.junit.Before;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Map;
import java.util.Random;
import java.util.UUID;
```

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

public class POSTRESTfulHappyTest {
    static Random random = new Random();
    int randomInt = random.nextInt(1000000) + 1;
    String KEY = randomInt + "-" + "AUTO";
    protected static final Logger logger =
        LoggerFactory.getLogger(POSTRESTfulHappyTest.class);
    public String EMAIL = "auto@gmail.com";
    public String TYPE = "AUTO";
    public String GUID = String.valueOf(UUID.randomUUID());
    public int NO = random.nextInt(100000) + 1;
    public long AMOUNT = 888;
    public int R_NO = random.nextInt(10000) + 1;
    public String REC_TYPE = "AUTO";
    public String TR_TYPE = "AUTO";
    public String METHOD = "AUTO";
    public String NAME = "Auto";
    public String requestUri;
    public String requestBody;
    RESTfulAPIModel responseMessage;
    RestAssuredConnector connector;
    DBConnections dbConn;
    public String QUERY = "SELECT * FROM table.column
    where key = ?";


    @Before
    public void setupData() {
        RESTfulAPIModel requestMessageBody = new RESTfulAPIModel();
        dbConn = new DBConnections();
        requestUri = ConfigUtils.getProperty(ConfigUtils.
        ConfigKeys.BASE_URL_QA.toString())
        +
        ConfigUtils.getProperty(ConfigUtils.
        ConfigKeys.MIDDLE_URL_QA.toString())
```

```
        +
        ConfigUtils.getProperty(ConfigUtils.
        ConfigKeys.END_URL_QA.toString());

        requestBody = requestMessageBody.
        createModel(EMAIL, TYPE, GUID, NO, AMOUNT, R_NO, REC_TYPE, TR_TYPE,
            KEY, METHOD, NAME);
        responseMessage = new RESTfulAPIsModel();
        connector = new RestAssuredConnector();
    }
}
```

2. RESTful API - DB integration test steps:

- Make call to RESTful API having the payload generated in data modeling from **RESTfulAPIDataModel** class;
- Parse and validate response message and store the key;
- Connect to MySQL database;
- Query the records and filter by key stored;
- Validate the fields.

RESTful API - DB integration test implementation:

```
package financial.RESTfulAPIsDB.micro_db;

import io.restassured.response.Response;
import financial.db.DBConnections;
import financial.restfulapis.RESTfulAPIsDBPaymentModel;
import financial.utils.general.ConfigUtils;
import financial.utils.connector.RestAssuredConnector;
import org.junit.Before;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.util.Map;
import java.util.Random;
import java.util.UUID;

import static org.junit.Assert.assertEquals;

public class RESTfulAPIsDBValidationsHappyTest {

protected static final Logger logger =
    LoggerFactory.getLogger(RESTfulAPIsDBValidationsHappyTest.
class);
     static Random random = new Random();
    int randomInt = random.nextInt(1000000) + 1;
    String KEY = randomInt + "-" + "AUTO";
    public String EMAIL = "auto@gmail.com";
    public String TYPE = "AUTO";
    public String GUID = String.valueOf(UUID.randomUUID());
    public int NO = random.nextInt(100000) + 1;
    public long AMOUNT = 888;
    public int R_NO = random.nextInt(10000) + 1;
    public String REC_TYPE = "AUTO";
    public String TR_TYPE = "AUTO";
    public String METHOD = "AUTO";
    public String NAME = "Auto";
    public String requestUri;
    public String requestBody;
    RESTfulAPIsDBPaymentModel responseMessage;
    RestAssuredConnector connector;
    DBConnections dbConn;
    public String QUERY = "SELECT * FROM table where key = ?";

    @Before
    public void setupData() {
        RESTfulAPIDataModel requestMessageBody = new RESTfulAPIDataModel();
        dbConn = new DBConnections();
        requestUri = ConfigUtils.getProperty(ConfigUtils.
```

```
    ConfigKeys.BASE_URL_QA.toString())
    +
    ConfigUtils.getProperty(ConfigUtils.
    ConfigKeys.MIDDLE_URL_QA.toString())
    +
    ConfigUtils.getProperty(ConfigUtils.
    ConfigKeys.END_URL_QA.toString());

    requestBody = requestMessageBody.
    createModel(EMAIL, TYPE, GUID, NO, AMOUNT, R_NO, REC_TYPE, TR_TYPE,
        KEY, METHOD, NAME);
    responseMessage = new RESTfulAPIDataModel();
    connector = new RestAssuredConnector();
}

@Test
public void postRESTfulAPIDBvalidationsHappyTest() throws SQLException {
    RestAssuredConnector connector = new RestAssuredConnector();
    Map<String, String> headers =
        RestAssuredConnector.setHeaders(ConfigUtils.
    getProperty(ConfigUtils.ConfigKeys.
    TOKEN_RESTfulAPIsDB.
    toString()));
    Response RESTfulAPIsDBResponse = connector.postRequest(requestUri,
        headers, requestBody);

    logger.info("RESTful API URL:" + " " + requestUri);
    logger.info("requestBody:" + " " + requestBody);
    logger.info("Response:" + " " +
        RESTfulAPIsDBResponse.getBody().asString());

    assertEquals(RESTfulAPIsDBResponse.getStatusCode(), 200);
    int key = responseMessage.savePaymentKey(
    RESTfulAPIsDBResponse);
    ResultSet rs = dbConn.dbConn(ConfigUtils.
    getProperty(ConfigUtils.
    ConfigKeys.RESTfulAPIsDB_DB_URL_QA.name()), ConfigUtils.getProperty(
```

```
        String.valueOf(ConfigUtils.
        ConfigKeys.RESTfulAPIsDB_DB_USERNAME_QA)), ConfigUtils.getProperty(
        String.valueOf(ConfigUtils.
        ConfigKeys.DB_PASSWORD_QA)), QUERY, key);


        if (rs.next()) {
            int keyDB = rs.getInt("key");
            String noDB = rs.getString("no");
            String rNDBo = rs.getString("r_no");
            long amountDB = rs.getLong("amount");
            String emailDB = rs.getString("email");
            String cIdDB = rs.getString("c_id");
            int confDB = rs.getInt("conf");
            String guidDB = rs.getString("guid");
            String dateDB = rs.getString("date");
            String nameDB = rs.getString("name");
            if (keyDB == (key)) {
                assertEquals(amountDB, amount);
                logger.info("DB Validations for key: " + keyDB + " " +
                    "amount: " + amountDB + "email: " + emailDB + "name: " +
                    namme + "GUID: " + guidDB + "passed successfully");
            } else {
                logger.info("No key was found into the db");
            }
            dbConn.closeConnection(rs.getStatement().
            getConnection());
        }
    }
}
```

What we did not cover until now is **pom file**. As we are already familiarised with the scope of this file, classes and resources from pre-built libraries imports into the project for: **Kafka Producer**, **Kafka Consumer**, **Database**, RESTful APIs (we covered it in **Chapter 4**: **"Installing necessary tools and libraries"**) are recognized and can be properly used.

Kafka maven dependencies: Kafka Client, Kafka Serializer Avro Schema, Kafka Confluent Cloud required in our project to run successfully:

```
<!-- All necessary Kafka dependencies -->
   <dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-clients</artifactId>
        <version>${kafka.clients.version}</version>
   </dependency>

   <!-- https://mvnrepository.com/artifact/org.apache.
    kafka/kafka-server-common -->
   <dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-server-common</artifactId>
       <version>${kafka-server-common.version}
       </version>
       <scope>runtime</scope>
  </dependency>

   <dependency>
       <groupId>org.apache.avro</groupId>
       <artifactId>avro</artifactId>
       <version>${avro.version}</version>
   </dependency>

   <!-- https://mvnrepository.com/artifact/io.confluent
       /kafka-avro-serializer -->
   <dependency>
       <groupId>io.confluent</groupId>
       <artifactId>kafka-avro-serializer
       /artifactId>
       <version>${kafka-avro-serializer.version}</version>
    </dependency>

<dependency>
  <groupId>io.confluent</groupId>
```

```
 <artifactId>kafka-schema-registry-client</artifactId>
 <version>${kafka-schema-registry-client.version}
 </version>
</dependency>

    <!-- https://mvnrepository.com/artifact/io.confluent
    /kafka-schema-serializer -->
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-schema-serializer</artifactId>
        <version>${kafka-schema-serializer.version}
        </version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.
    kafka/kafka-server-common -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-server-common</artifactId>
        <version>${kafka-server-common.version}
        </version>
        <scope>runtime</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.
    kafka/kafka-clients -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>${kafka-clients.version}</version>
        </dependency>

   <!-- https://mvnrepository.com/artifact/io.confluent
    /kafka-json-serializer -->
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-json-serializer</artifactId>
```

```
        <version>${kafka-json-serializer.version}
        </version>
    </dependency>
```

Database maven dependencies: MySQL, PostgreSQL required in our project to run successfully:

```
  <!-- https://mvnrepository.com/artifact/com.mysql
  /mysql-connector-j -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>${mysql-connector-j.version}</version>
    </dependency>

 <!-- https://mvnrepository.com/artifact/org.postgresql/
        postgresql -->
   <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
         <version>${postgresql.version}</version>
   </dependency>
```

runners execute tests defined with JUnit test classes, by iterating through all the tests defined within a test class, executing them sequentially. JUnit tests are grouped by test suites and projects for a better organization and execution.

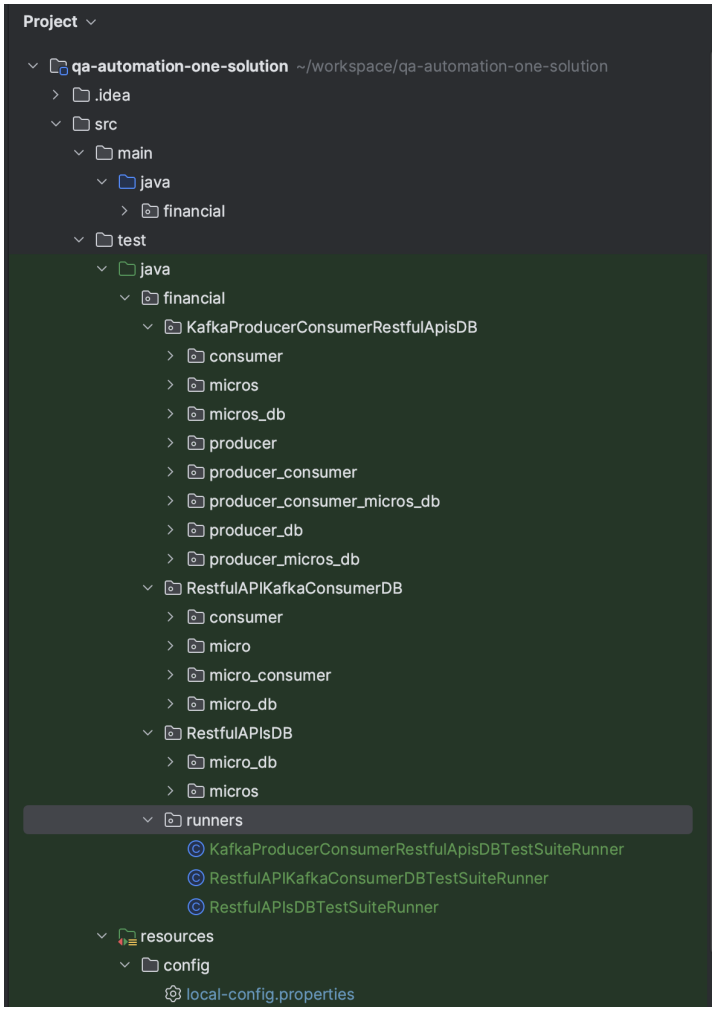

Figure 6.7: runners

runners test results:

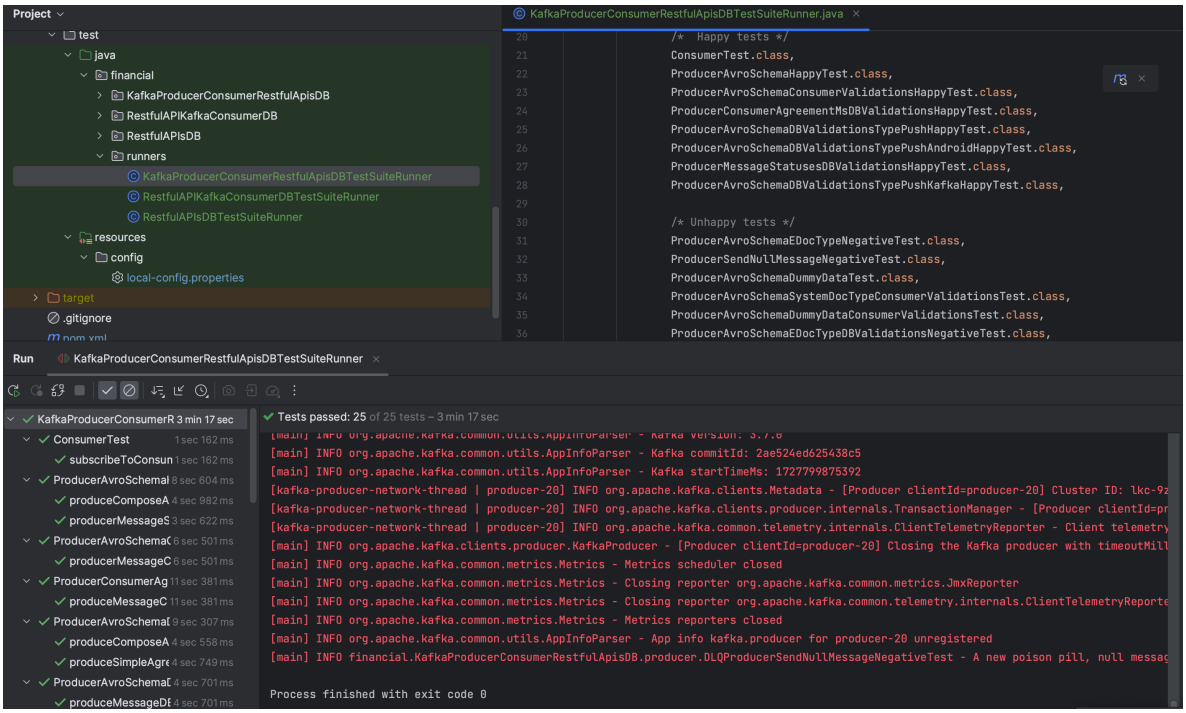

Figure 6.8: RESTful APIs Kafka DB Runner Test Results

compile phase for the project reflecting that the build is with success.

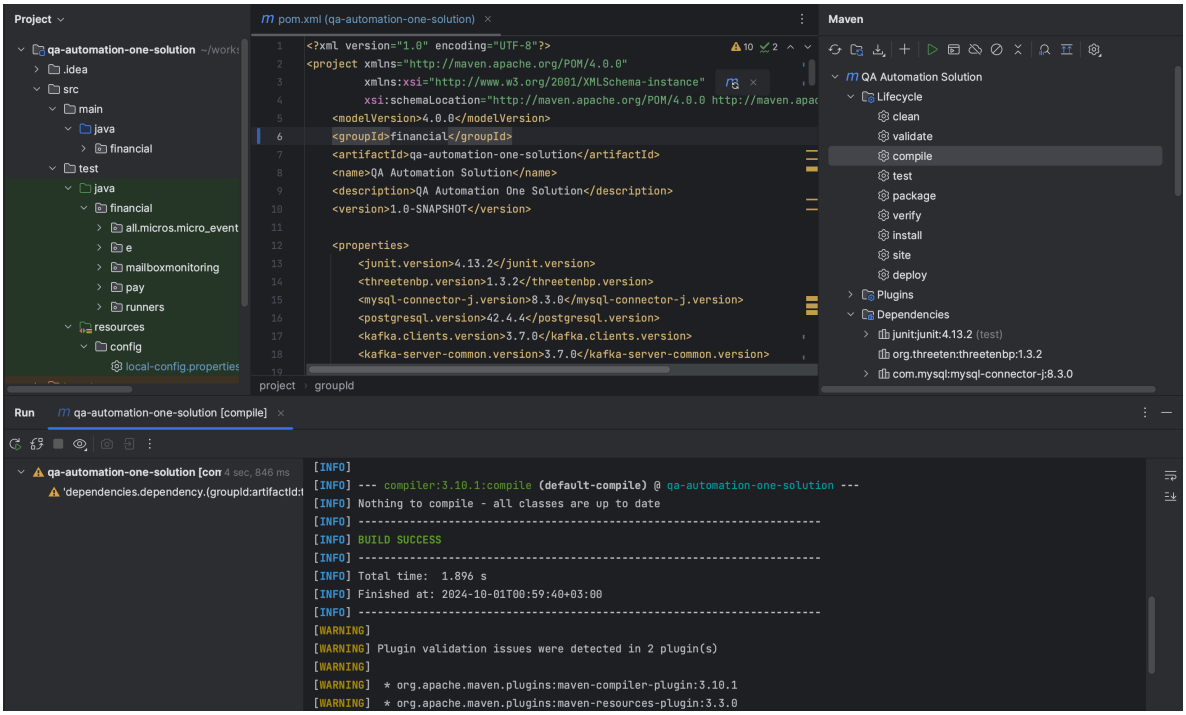

Figure 6.9: maven compile phase BUILD SUCCESS

test phase running all the JUnit tests:

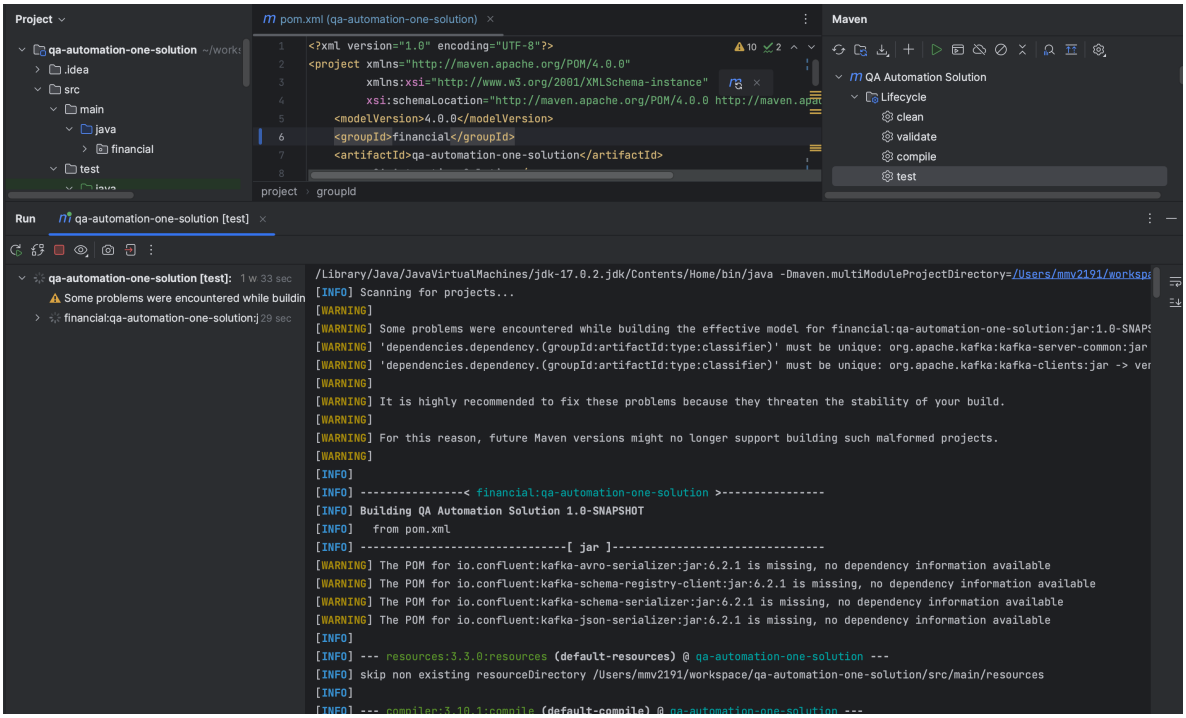

Figure 6.10: maven test phase

BUILD FAILURE in test phase having test failures:

```
[main] INFO org.apache.kafka.clients.consumer.internals.ConsumerCoordinator - [Consumer clientId=consumer-automation_sa
[main] INFO org.apache.kafka.clients.consumer.internals.ConsumerCoordinator - [Consumer clientId=consumer-automation_sa
[main] INFO org.apache.kafka.common.metrics.Metrics - Metrics scheduler closed
[main] INFO org.apache.kafka.common.metrics.Metrics - Closing reporter org.apache.kafka.common.metrics.JmxReporter
[main] INFO org.apache.kafka.common.metrics.Metrics - Closing reporter org.apache.kafka.common.telemetry.internals.Clie
[main] INFO org.apache.kafka.common.metrics.Metrics - Metrics reporters closed
[main] INFO org.apache.kafka.common.utils.AppInfoParser - App info kafka.consumer for consumer-automation_sales_events_
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 s - in financial.RestfulAPIKafkaConsumerDB
[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   ProducerAvroSchemaMicrosDBValidationsTypePushHappyTest.sendKafkaMessageDbValidationsHappyTest:118 expected:<"
[ERROR]   DeleteRestfulApiHappyTest.deleteMMPayPaymentHappyTest:32 expected:<200> but was:<400>
[ERROR]   GETRestfulApiHappyTest.getAgreementCustomerHappyTest:34 expected:<200> but was:<400>
[ERROR] Errors:
[ERROR]   ProducerAvroSchemaEDocTypeMicrosDBValidationsTypePushNegativeTest.sendKafkaMessageDbValidationsHappyTest:118
[ERROR]   POST_GET_DELETE_RestfulApiValidationsTest.postGetDeleteMMPayPaymentHappyTest:82 » URISyntax Illegal character
[INFO]
[ERROR] Tests run: 48, Failures: 3, Errors: 2, Skipped: 0
[INFO]
```

Figure 6.11: BUILD FAILURE TEST

verify phase:

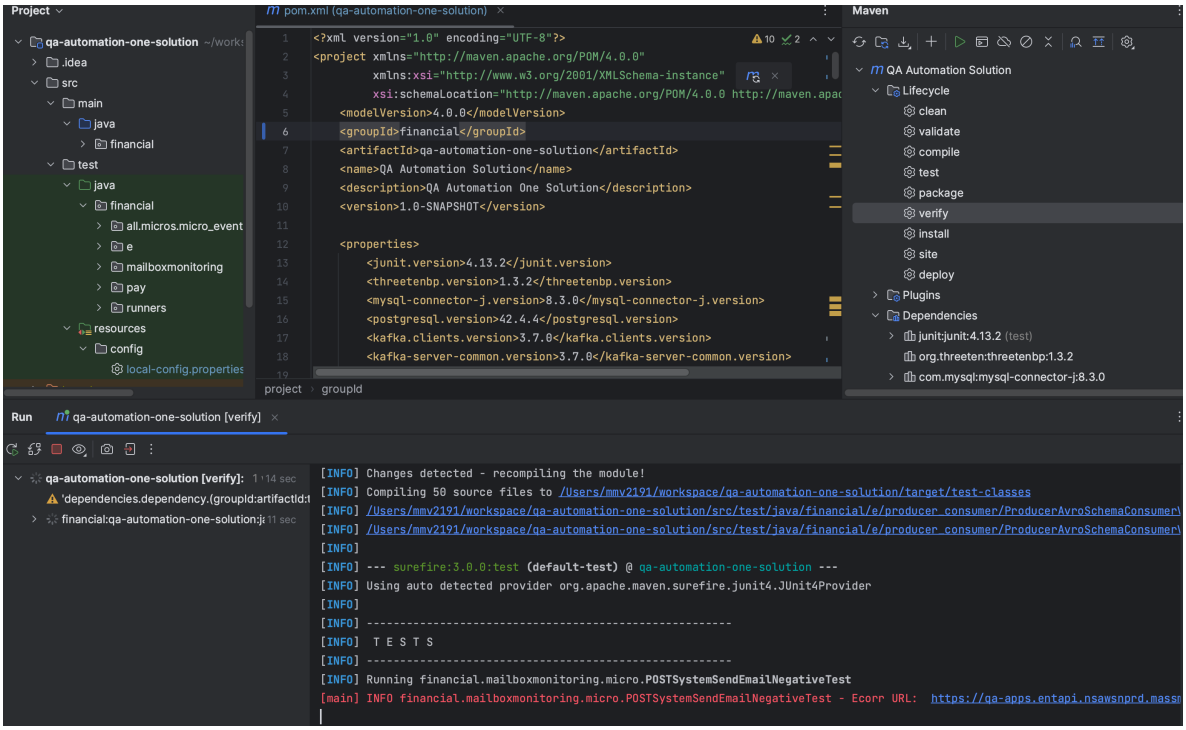

Figure 6.12: maven verify phase

maven surefire report test results txt files with logs for each test are stored, tests can be investigated when are resulting into failures.

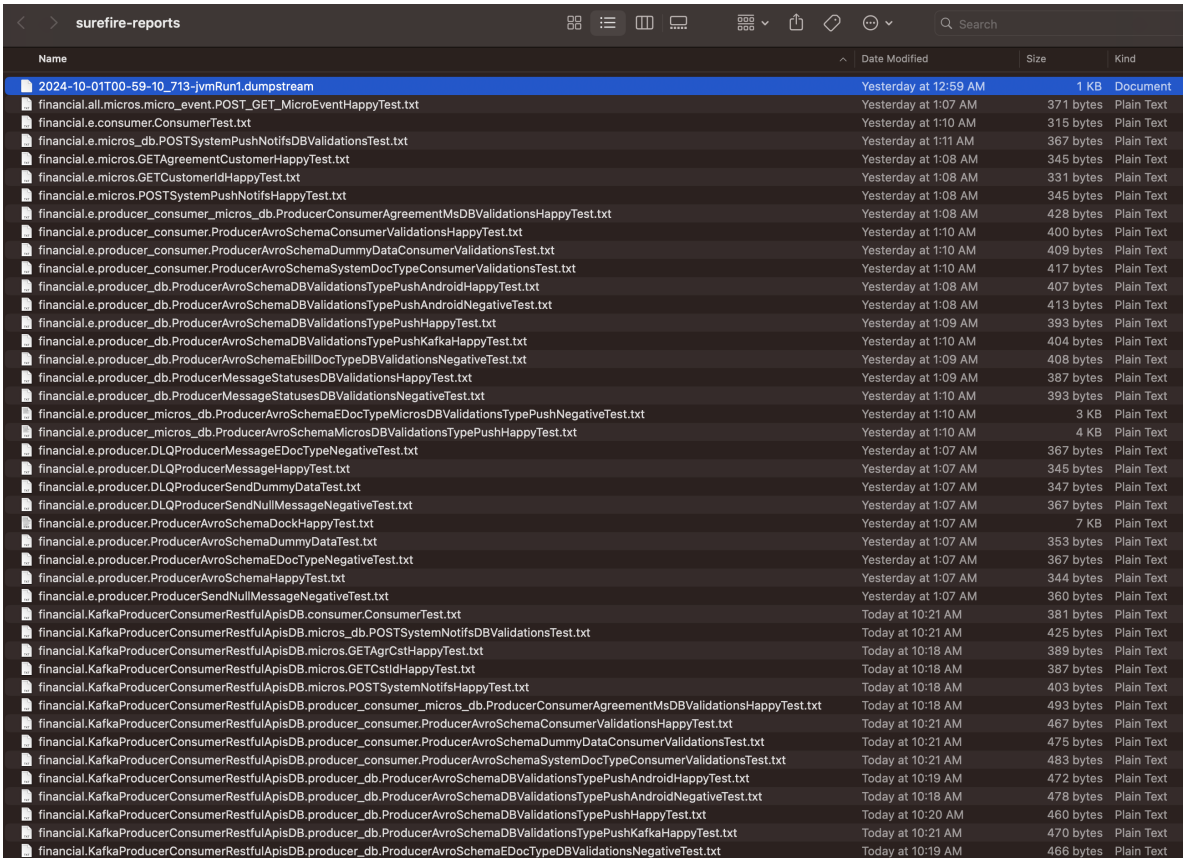

Figure 6.13: maven surefire test results report

- **Lessons learned and best practices**

It's very difficult to establish exactly what are the best practices, due to many factors: team capacity and skill, project technologies, project complexity, deadlines, project capabilities, budgets.

After all my work experience and projects I've been working on I could set some important key aspects to have a successful automation testing solution in place:

- simplicity of the framework to meet the project's requirements;
- design tests in the scope of functional, integration, end-to-end and regression testing;
- design patterns: Factory, Builder, Page Object Model, Singleton, Template, Data-Driven-Test, Page Object Model (POM);
- testing frameworks: JUnit or TestNG;
- organized tests using xml file test suites for TestNG and runners for JUnit tests;
- autonomously implementation tests by creating new test data and not rely on existent one;
- use assertions to validate data (actual data = expected data);
- focus on test coverage in implementing the important use cases and the edge case;
- run project phases to make sure that builds are successful;
- logging information in our tests for more detailed information about the tests' progress, debugging and troubleshooting purposes;
- git flow branches practices;
- naming conventions: classes, methods relevant name, test name should match branch name and Jira id;
- config. files to store urls, test data, configurations;
- development, staging and production environments set up;
- build control tool such as Apache Maven;
- integrate automation testing into CI/CD;

- configure jobs for smoke, end-to-end and regression testing to run periodically, ideally smoke after each development build, end-to-end daily.

Chapter 7

Part 4: Overcoming Common Challenges

- **Chapter 9: Addressing Common Issues in Automation Testing**

1. ○ **Tips for troubleshooting and debugging.**

Automation testing solutions could be challenging sometimes. Some challenges encountered along the way:

- product software must be stable and functional in order to have automation implemented;
- complexity and technologies could create some challenges in automating it;
- high test failures percentage due to timing issue, unstable environments;
- development frequent changes in the code that impact the testing solution;
- test coverage low percentage in automation to identify important use cases not being covered could result in framework inefficiency;
- false positive and negative test results;

Solutions:

- stable environments set up;
- implement clear, autonomous, reliable tests;
- update and maintain tests;
- increase test coverage percentage;
- implement tests to cover happy and unhappy paths;
- implement tests in scope of functional, integration, end-to-end and regression testing by generating new test data.

Tips for debugging and troubleshooting automated tests.

Is a challenging task, depending on many factors from complexity to system configuration. Might be a complex custom situation and might depend on a specific context, a lot of things could go wrong: incorrect configurations, services down, database and network latency are just a few issues that we can't control in our tests and are difficult to debug. Is better to have a good understanding of the system architecture, setup and configurations to identify the root cause easier.

- run a test locally to identify the possible root cause;
- logs and errors could be very useful to understand the progress of test and the point didn't work;
- data used is also very important if is the correct one;
- debug code by setting breakpoints and inspecting the variables;
- use assertions to do validations and check the actual data meet the expected data;
- check reports with test results;
- check documentation, dedicated communities for automation testing.