REST ASSURED NOTES – BASICS

Framework Design

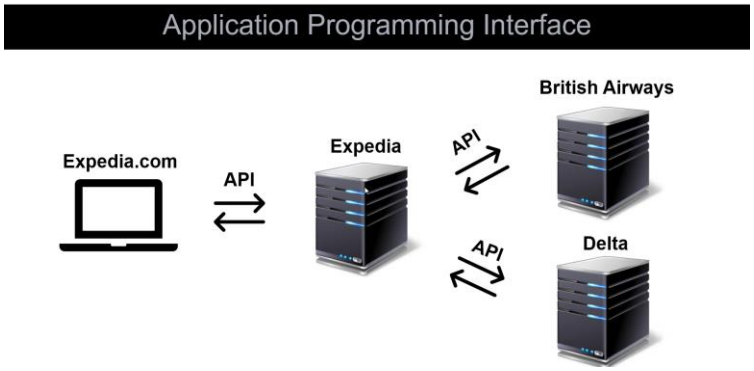
Goals:

- ✓ Scalable and extensible
- ✓ Reusable Rest Assured specifications
- ✓ Reusable Rest Assured API requests
- ✓ Separation of API layer from test layer
- ✓ POJOs for Serialization and Deserialization
- ✓ Singleton Design Pattern
- ✓ Lombok for reducing Boilerplate code
- ✓ Builder pattern for Setter methods in POJOs
- ✓ Robust reporting and logging using Allure
- ✓ Automate positive and negative scenarios
- ✓ Support parallel execution
- ✓ Data driven using TestNG Data Provider
- ✓ Automated access token renewal
- ✓ Maven command line execution
- ✓ Integration with Git
- ✓ Integration with Jenkins

Tools and Technologies

- Rest Assured
- TestNG
- Java
- Allure Reports
- Hamcrest
- Jackson API
- Lombok
- GitHub
- Jenkins

REST API (Representational State Server) :



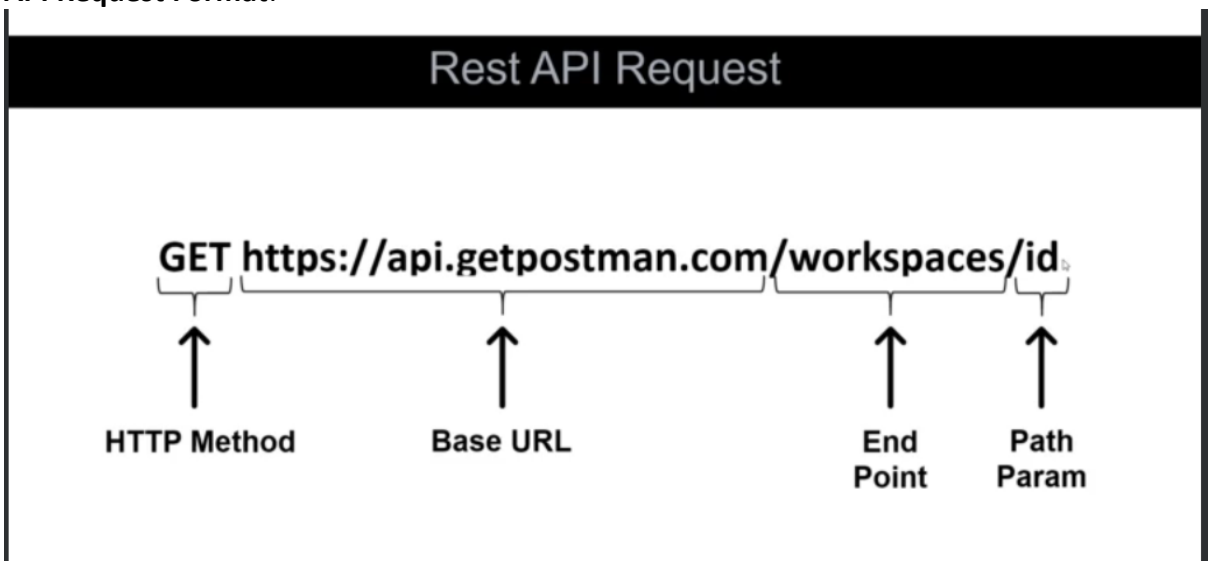
REST 6 Constraints:

- *Client Server,*
- *Stateless,*
- *Cache,*
- *Uniform Interface,*
- *Layered System,*
- *Code on Demand.*

JSON (Javascript Object Notation). It is lightweight, human readable, easy , key-value pairs.

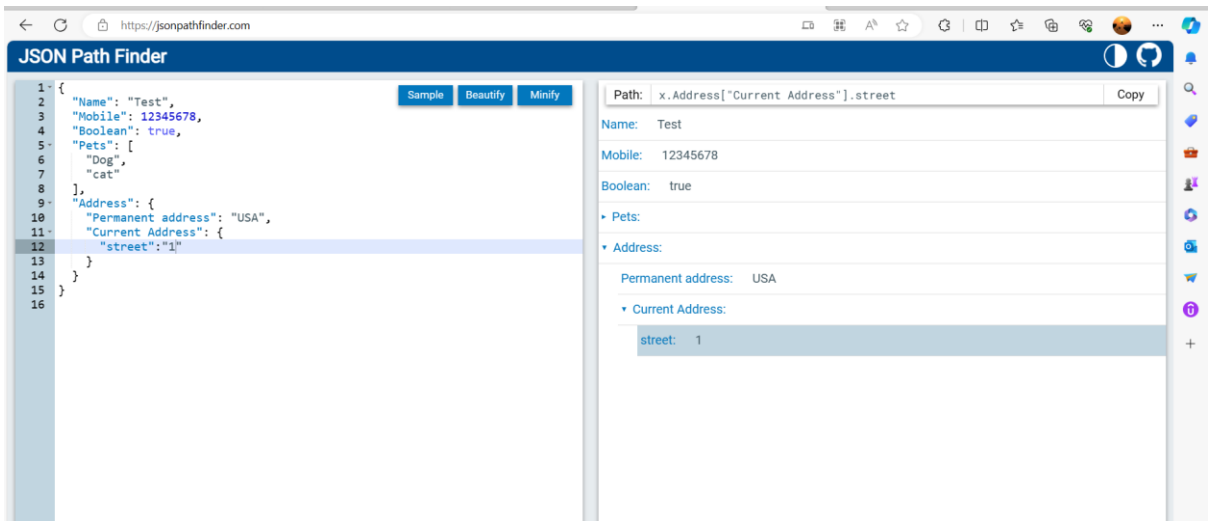
```
{
  "Name": "Test",
  "Mobile": 12345678,
  "Boolean": true,
  "Pets": [
    "Dog",
    "cat"
  ],
  "Address": {
    "Permanent address": "USA",
    "current Address": "AU"
  }
}
```

API Request Format:



JSON Path:
Groovy GPath , Jayway JsonPath

[JSON Path Finder]



x.Pets[0]
x.Address["Permanent address"]

HTTP Methods – RFC 7231/RFC 5789

Method	Description	Request body	Response body	Safe	Idempotent	Cacheable
GET	Transfer a current representation of the target resource	No	Yes	Yes	Yes	Yes
HEAD	Same as GET, but only transfer the status line and header section	No	No	Yes	Yes	Yes
POST	Perform resource-specific processing on the request payload	Yes	Yes	No	No	In some cases
PUT	Replace all current representations of the target resource with the request payload	Yes	No	No	Yes	No
DELETE	Remove all current representations of the target resource	Optional	Optional	No	Yes	No
CONNECT	Establish a tunnel to the server identified by the target resource	No	Yes	No	No	No
OPTIONS	Describe the communication options for the target resource	No	Yes	Yes	Yes	No
TRACE	Perform a message loop-back test along the path to the target resource	No	No	Yes	Yes	No
PATCH	Perform partial modification of the target resource	Yes	Yes	No	No	No

REST ASSURED: GETTING STARTED:

Static Imports -> Readability, Reduced Lines:
With the help of Java static import, we can access the static members of a class directly without class name or any object

```java
package com.rest;

import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.testng.Assert;
import org.testng.annotations.Test;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import static io.restassured.RestAssured.given;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;
```

```java
@Test
public void validate_response_body(){
    given().
            baseUri( s: "https://api.postman.com").
            header( s: "X-Api-Key",  o: "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2").
    when().
            get( s: "/workspaces").
    then().
            log().all().
            assertThat().
            statusCode( i: 200).
            body( s: "workspaces.name", hasItems("Team Workspace", "My Workspace", "My Workspace2"),
                    ...objects: "workspaces.type", hasItems("team", "personal", "personal"),
                    "workspaces[0].name", equalTo( operand: "Team Workspace"),
                    "workspaces[0].name", is(equalTo( operand: "Team Workspace")),
                    "workspaces.size()", equalTo( operand: 3),
                    "workspaces.name", hasItem("Team Workspace")
            );
}

@Test
public void validate_response_body_hamcrest_l
    given().
            baseUri( s: "https://api.postman.co
```

```java
org.hamcrest.Matchers

@NotNull
@Contract("_->new")
public static <T> org.hamcrest.Matcher<Iterable<? super T>> hasItem(
    T item
)
```

Maven: org.hamcrest:hamcrest:2.2 (hamcrest-2.2.jar)

Request and Response Specification with Post Request :
Sample Post Spec:

```
public class AutomatePost {

    @BeforeClass
    public void beforeClass(){
        RequestSpecBuilder requestSpecBuilder = new RequestSpecBuilder().
                setBaseUri("https://api.postman.com").
                addHeader( headerName: "X-Api-Key",  headerValue: "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2").
                setContentType(ContentType.JSON).
                log(LogDetail.ALL);
        RestAssured.requestSpecification = requestSpecBuilder.build();

        ResponseSpecBuilder responseSpecBuilder = new ResponseSpecBuilder().
                expectStatusCode( expectedStatusCode: 200).
                expectContentType(ContentType.JSON).
                log(LogDetail.ALL);
        RestAssured.responseSpecification = responseSpecBuilder.build();
    }
```

Sample Test in BDD (Given,When,Then) :

```
    @Test
    public void validate_post_request_bdd_style(){
        String payload = "{\n" +
                "    \"workspace\": {\n" +
                "        \"name\": \"myFirstWorkspace\",\n" +
                "        \"type\": \"personal\",\n" +
                "        \"description\": \"Rest Assured created this\"\n" +
                "    }\n" +
                "}";
        given().
                body(payload).
        when().
                post( s: "/workspaces").
        then().
                log().all().
                assertThat().
                body( s: "workspace.name", equalTo( operand: "myFirstWorkspace"),
                        ...objects: "workspace.id", matchesPattern( regex: "^[a-z0-9-]{36}$"));
    }
```

Sample Test in non BDD : From Request Specification pre-configured.

```
@Test
public void validate_post_request_non_bdd_style(){
    String payload = "{\n" +
            "    \"workspace\": {\n" +
            "        \"name\": \"myFirstWorkspace2\",\n" +
            "        \"type\": \"personal\",\n" +
            "        \"description\": \"Rest Assured created this\"\n" +
            "    }\n" +
            "}";

    Response response = with().
            body(payload).
            post( s: "/workspaces");
    assertThat(response.<~>path( s: "workspace.name"), equalTo( operand: "myFirstWorkspace2"));
    assertThat(response.<~>path( s: "workspace.id"), matchesPattern( regex: "^[a-z0-9-]{36}$"));
}
```

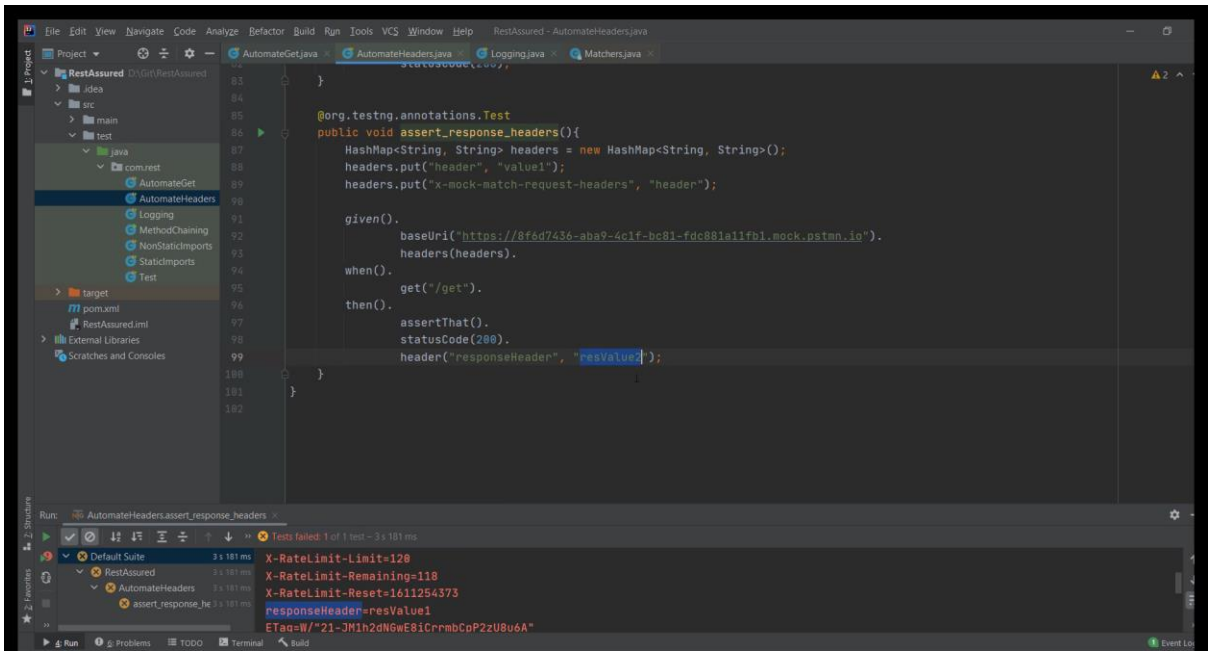
Sample Test by passing request body from file. We can also pass values as Collection object such as Map.

```
@Test
public void validate_post_request_payload_from_file(){
    File file = new File( pathname: "src/main/resources/CreateWorkspacePayload.json");
    given().
            body(file).
    when().
            post( s: "/workspaces").
    then().
            log().all().
            assertThat().
            body( s: "workspace.name", equalTo( operand: "mySecondWorkspace"),
                    ...objects: "workspace.id", matchesPattern( regex: "^[a-z0-9-]{36}$"));
}
```

Rest Assured Enable Logging:

```
@org.testng.annotations.Test
public void request_response_logging(){
    given().
            baseUri("https://api.postman.com").
            header("X-Api-Key", "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2").
            log().all().
    when().
            get("/workspaces").
    then().
            log().all().
            assertThat().
            statusCode(200);
}
}
```

Rest Assured Can Assert Response Headers:



Multiple Response Headers Assertion:

```
    @org.testng.annotations.Test
    public void assert_response_headers(){
        HashMap<String, String> headers = new HashMap<String, String>();
        headers.put("header", "value1");
        headers.put("x-mock-match-request-headers", "header");

        given().
                baseUri("https://8f6d7436-aba9-4c1f-bc81-fdc881a11fb1.mock.pstmn.io").
                headers(headers).
        when().
                get("/get").
        then().
                assertThat().
                statusCode(200).
                headers("responseHeader", "resValue1",
                        "X-RateLimit-Limit", "120");
    }
}
```

Extract All and Print Response Headers:
Also, We can do the multi value headers too.

```
Headers extractedHeaders = given().
        baseUri("https://8f6d7436-aba9-4c1f-bc81-fdc881a11fb1.mock.pstmn.io").
        headers(headers).
  when().
        get("/get").
  then().
        assertThat().
        statusCode(200).
        extract().
        headers();

  for(Header header: extractedHeaders){
      System.out.print("header name = " + header.getName() + ", ");
      System.out.println("header value = " + header.getValue());
  }
```

Request Specification in Rest Assured:

These can be provided in upfront in before class and assign to static variable of RequestSpecification from Rest Assured.

Later, all the requests can be triggered without providing base urls, headers and other repetitive values.

```
public class RequestSpecificationExample {

    @BeforeClass
    public void beforeClass(){
/*      requestSpecification = with().
            baseUri("https://api.postman.com").
            header("X-Api-Key", "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2").
            log().all();*/
        RequestSpecBuilder requestSpecBuilder = new RequestSpecBuilder();
        requestSpecBuilder.setBaseUri("https://api.postman.com");
        requestSpecBuilder.addHeader("X-Api-Key", "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2");
        requestSpecBuilder.log(LogDetail.ALL);

        RestAssured.requestSpecification = requestSpecBuilder.build();
    }

    @Test
    public void validate_status_code(){
        Response response = get("/workspaces").then().log().all().extract().response();
        assertThat(response.statusCode(), is(equalTo(200)));
    }

    @Test
    public void validate_response_body(){
        Response response = get("/workspaces").then().log().all().extract().response();
        assertThat(response.statusCode(), is(equalTo(200)));
        assertThat(response.path("workspaces[0].name").toString(), equalTo("Team Workspace"));
    }
}
```

If we want to know what are the request specifications configured in before test, then we have a way to query it in rest assured from class "QueryableRequestSpecification"

```java
public class RequestSpecificationExample {

    @BeforeClass
    public void beforeClass(){
/*      requestSpecification = with().
            baseUri("https://api.postman.com").
            header("X-Api-Key", "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2").
            log().all();*/
        RequestSpecBuilder requestSpecBuilder = new RequestSpecBuilder();
        requestSpecBuilder.setBaseUri("https://api.postman.com");
        requestSpecBuilder.addHeader("X-Api-Key", "PMAK-5ff2d720d2a39a004250e5da-c658c4a8a1cee3516762cb1a51cba6c5e2");
        requestSpecBuilder.log(LogDetail.ALL);

        RestAssured.requestSpecification = requestSpecBuilder.build();
    }

    @Test
    public void queryTest(){
        QueryableRequestSpecification queryableRequestSpecification = SpecificationQuerier.
                query(RestAssured.requestSpecification);
        System.out.println(queryableRequestSpecification.getBaseUri());
        System.out.println(queryableRequestSpecification.getHeaders());
    }

    @Test
    public void validate_status_code(){
        Response response = get("/workspaces").then().log().all().extract().response();
        assertThat(response.statusCode(), is(equalTo(200)));
    }
```

Rest-Assured : Response Specification:

Add common response specification in before class and use them in test as "then().spec(responseSpecObject)"

```java
Analyze  Refactor  Build  Run  Tools  VCS  Window  Help    RestAssured - ResponseSpecificationExample.java
ResponseSpecificationExample.java
        requestSpecBuilder.log(LogDetail.ALL);

        RestAssured.requestSpecification = requestSpecBuilder.build();

/*        responseSpecification = RestAssured.expect().
                statusCode(200).
                contentType(ContentType.JSON).
                log().all();*/

        ResponseSpecBuilder responseSpecBuilder = new ResponseSpecBuilder().
                expectStatusCode(200).
                expectContentType(ContentType.JSON).
                log(LogDetail.ALL);
        responseSpecification = responseSpecBuilder.build();
    }

    @Test
    public void validate_status_code(){
        get("/workspaces").
        then().spec(responseSpecification);
    }

    @Test
    public void validate_response_body(){
        Response response = get("/workspaces").
        then().spec(responseSpecification).
                        extract().
                        response();
        assertThat(response.path("workspaces[0].name").toString(), equalTo("Team Workspace"));
    }
```

Encoding and Decoding:

By default encoding added in rest assured is UTF-8. This may cause request failures if they are not configured properly.

https://github.com/rest-assured/rest-assured/wiki/Usage#encoder-config

RestAssured.config = RestAssured.config(config().encoderConfig(encoderConfig().defaultCharsetForContentType("UTF-16", "application/xml")));

Query Parameter:

We can use query parameters to control what data is returned in endpoint resources. It appears at the end of the URL after the question mark (?) and helps us to control the set of items and properties in responses, and the order of the items returned.
Consider the following GitHub API URL:
https://api.github.com/user/repos?sort=created&direction=desc
This will list all repositories for an authenticated user but the response properties will be sorted by repository created and in descending order.

```
@Test
public void multiple_query_parameters(){
    HashMap<String, String> queryParams = new HashMap<>();
    queryParams.put("foo1", "bar1");
    queryParams.put("foo2", "bar2");
    given().
            baseUri( s: "https://postman-echo.com").
            //          param("foo1", "bar1")
   //         queryParam("foo1", "bar1").
   //         queryParam("foo2", "bar2").
            queryParams(queryParams).
            log().all().
    when().
            get( s: "/get").
```

Path Parameter:

Path parameters are variables in a URL path. They are used to point to a specific resource within a collection. We can define multiple PATH parameters and each of them is represented by a curly brace {}.
Consider the following GitHub API url:
https://api.github.com/users/:username/repos
This will list all public repositories for the specified user with the value username.
:username is the Path parameter in the above url.

```
@Test
public void path_parameter(){
    given().
            baseUri( s: "https://reqres.in").
            pathParam( s: "userId",  o: "2").
            log().all().
    when().
            get( s: "/api/users/{userId}").
    then().
            log().all().
            assertThat().
            statusCode( i: 200);
}
```

Form Data:

Form-data is one of the formats for data sent from a web form. Specifically, it encodes values entered into a form as name-value pairs and sends them with the Content-Type header set to multipart/form-data. The main features of form-data include:

- *Ability to send not only text but also files.*
- *Ability to split and send the transmitted data into parts.*
- *Ability to specify the content type for each part.*

```
@Test
public void multipart_form_data(){
    given().
            baseUri( s: "https://postman-echo.com").
            multiPart( s: "foo1",  s1: "bar1").
            multiPart( s: "foo2",  s1: "bar2").
            log().all().
    when().
            post( s: "/post").
    then().
            log().all().
            assertThat().
            statusCode( i: 200);
}
```

Upload File using Multi-Part Form Data:

```
@Test
public void upload_file_multipart_form_data(){
    String attributes = "{\"name\":\"temp.txt\",\"parent\":{\"id\":\"123456\"}}";
    given().
            baseUri( s: "https://postman-echo.com").
            multiPart( s: "file", new File( pathname: "temp.txt")).
            multiPart( s: "attributes", attributes,  s2: "application/json").
            log().all().
    when().
            post( s: "/post").
    then().
            log().all().
            assertThat().
            statusCode( i: 200);
}
```

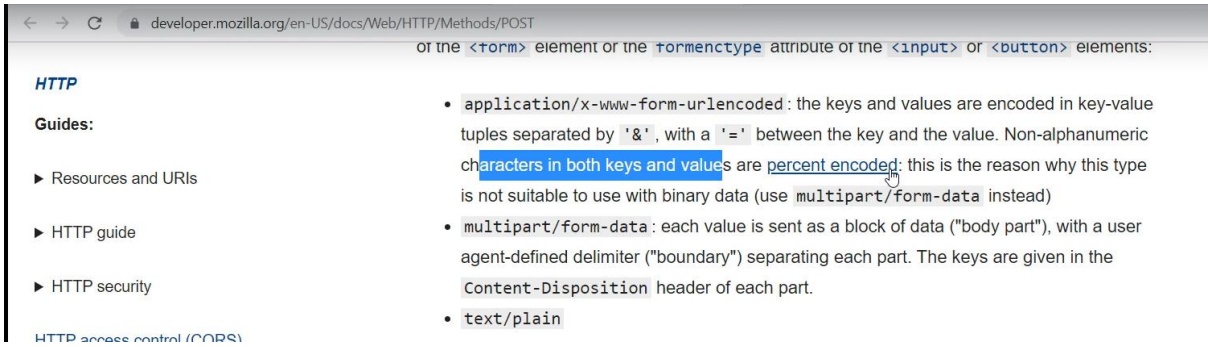
Download File using get request :

receive response as byte array or Input stream:

```
@Test
public void download_file() throws IOException {
    InputStream is = given().
            baseUri( s: "https://raw.githubusercontent.com").
            log().all().
    when().
            get( s: "/appium/appium/master/sample-code/apps/ApiDemos-debug.apk").
    then().
            log().all().
            extract().
            response().asInputStream();

    OutputStream os = new FileOutputStream(new File( pathname: "ApiDemos-debug.apk"));
    byte[] bytes = new byte[is.available()];
    is.read(bytes);
    os.write(bytes);
    os.close();
}
```

URL Encoded Form:

form data key value pairs provided in request with encoding as mentioned below



Percent Encoded Non-Alphanumeric :

Percent-encoding is a mechanism to encode 8-bit characters that have specific meaning in the context of URLs. It is sometimes called URL encoding. The encoding consists of substitution: A '%' followed by the hexadecimal representation of the ASCII value of the replace character.

Special characters needing encoding are: `':'`, `'/'`, `'?'`, `'#'`, `'['`, `']'`, `'@'`, `'!'`, `'$'`, `'&'`, `"'"`, `'('`, `')'`, `'*'`, `'+'`, `','`, `';'`, `'='`, as well as `'%'` itself. Other characters don't need to be encoded, though they could.

':'	'/'	'?'	'#'	'['	']'	'@'	'!'	'$'	'&'	"'"	'('	')'	'*'	'+'	','
%3A	%2F	%3F	%23	%5B	%5D	%40	%21	%24	%26	%27	%28	%29	%2A	%2B	%2C

```java
@Test
public void form_urlencoded(){
    given().
            baseUri("https://postman-echo.com").
            config(config().encoderConfig(EncoderConfig.encoderConfig()
                    .appendDefaultContentCharsetToContentTypeIfUndefined(false))).
            formParam("key1", "value1").
            formParam("key 2", "value 2").
            log().all().
    when().
            post("/post").
    then().
            log().all().
            assertThat().
            statusCode(200);
}
}
```

Validating JSON Schema from Rest Assured: External Library

Generally used to validate the json response data format and the mandatory fields, field type etc.

```
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;
import static io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchemaInClasspath;

public class JsonSchemaValidation {

    @Test
    public void validateJsonSchema() {
        given().
                baseUri( s: "https://postman-echo.com").
                log().all().
        when().
                get( s: "/get").
        then().
                log().all().
                assertThat().
                statusCode( i: 200).
                body(matchesJsonSchemaInClasspath( pathToSchemaInClasspath: "EchoGet.json"));
    }
}
```

Log Rest Assured Req/Res details to the Log File:

Useful in CI CD via PrintStream class. Also it has option to print what is necessary with Pretty print and other options.

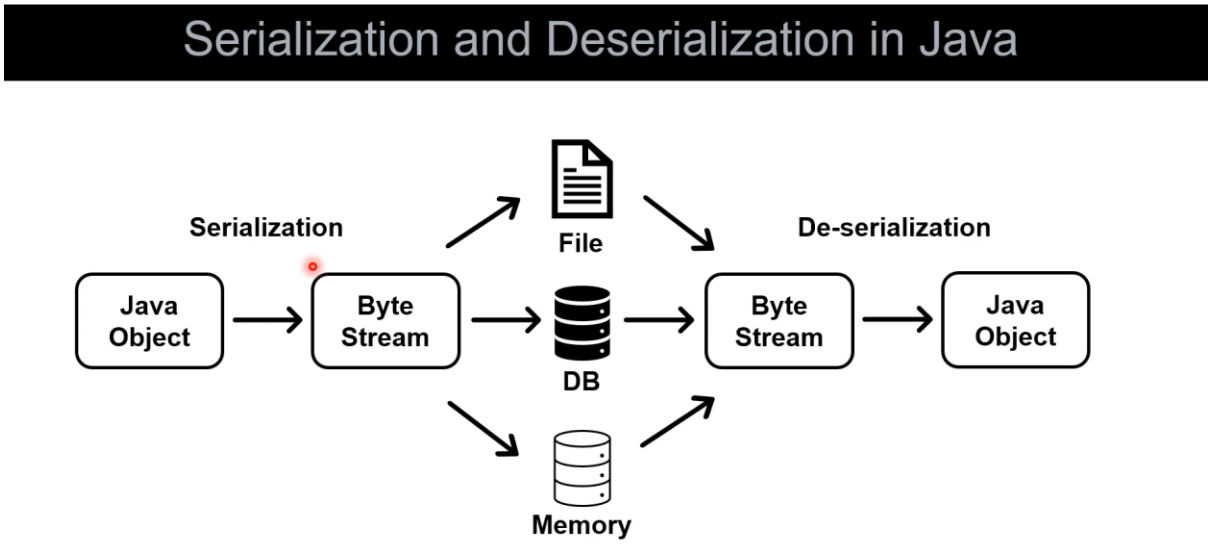
```
    @Test
    public void loggingFilter() throws FileNotFoundException {
        PrintStream FileOutPutStream = new PrintStream(new File("restAssured.log"));
        given().
                baseUri("https://postman-echo.com").
                filter(new RequestLoggingFilter(LogDetail.BODY, FileOutPutStream)).
                filter(new ResponseLoggingFilter(LogDetail.STATUS, FileOutPutStream)).
//                log().all().
        when().
                get("/get").
        then().
//                log().all().
                assertThat().
                statusCode(200);
    }
}
```

Provide Logging in Req/Res Specification builder class so it can be used again in another tests.

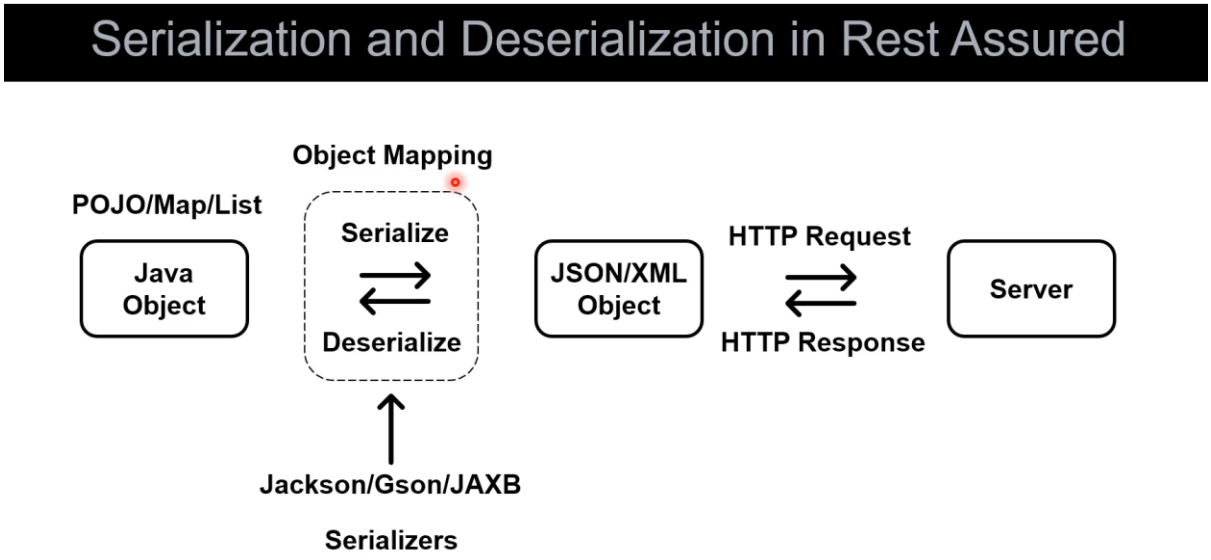
given should contain request Specification object and then spec should contain response specification

```
public class Filters {
    2 usages
    RequestSpecification requestSpecification;
    2 usages
    ResponseSpecification responseSpecification;

    @BeforeClass
    public void beforeClass() throws FileNotFoundException {
        PrintStream fileOutPutStream = new PrintStream(new File( pathname: "restAssured.log"));

        RequestSpecBuilder requestSpecBuilder = new RequestSpecBuilder().
                addFilter(new RequestLoggingFilter(fileOutPutStream)).
                addFilter(new ResponseLoggingFilter(fileOutPutStream));
        requestSpecification = requestSpecBuilder.build();

        ResponseSpecBuilder responseSpecBuilder = new ResponseSpecBuilder();
        responseSpecification = responseSpecBuilder.build();
    }

    @Test
    public void loggingFilter() throws FileNotFoundException {
        given(requestSpecification).
                baseUri( s: "https://postman-echo.com").
                log().all().
        when().
                get( s: "/get").
        then().spec(responseSpecification).
                log().all().
                assertThat().
                statusCode( i: 200);
```

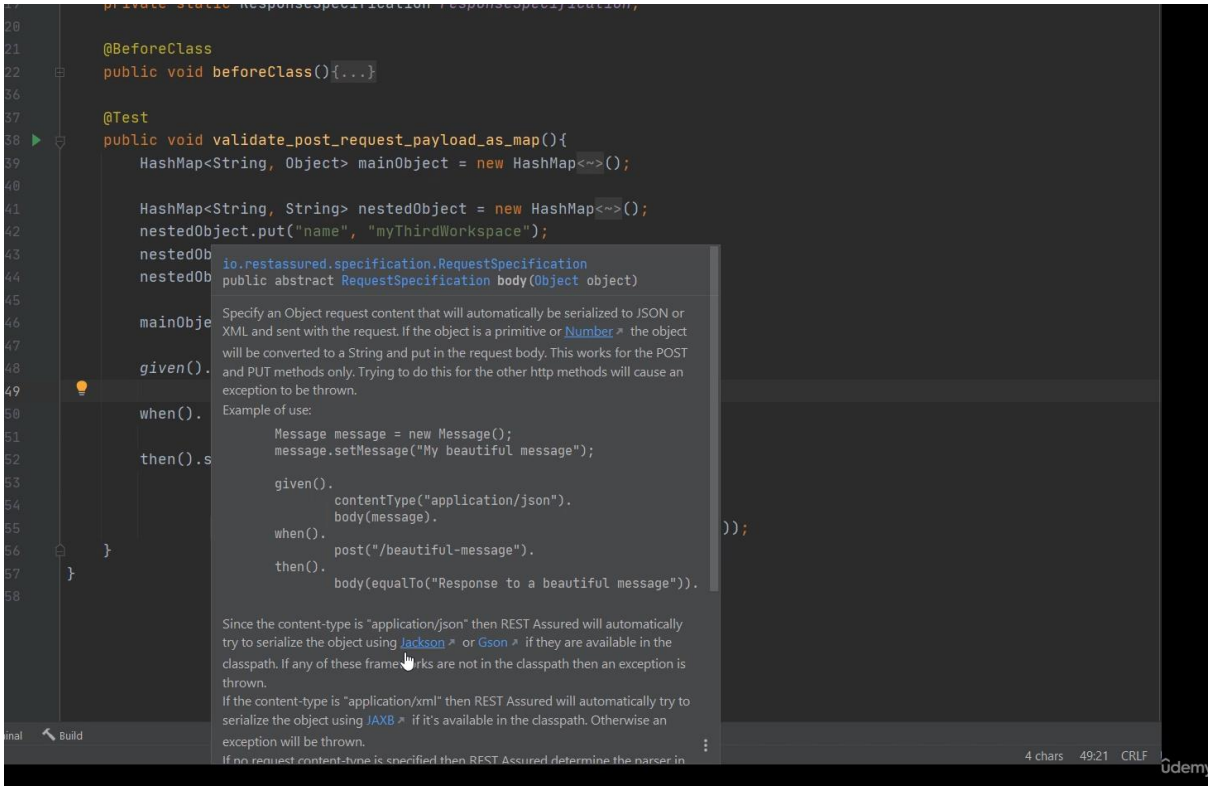
Serialization and De-Serialization in Java:



Rest Assured: *Object mapping of JSON/XML files into Java object either POJO/relevant class/collections and vice versa.*



Serialize Map to JSON Object using Jackson-databind Maven Dependency:
Method used to serialize java object to JSON here is writeValueAsString(javaObjects)



We can also create JSON file via Object-Mapper.

```
JSON Object:
=====================================
ObjectMapper
        CreateObjectNode() -> returns instance of ObjectNode class

ObjectNode
        put(String, T)

ObjectNode <->  HashMap

put(String, T) <-> put(String, T)

Get JSON Object as String -> writeValueAsString() from ObjectMapper
Get JSON Object -> writerWithDefaultPrettyPrinter() from ObjectMapper

RestAssured
        -> Pass String
        -> Pass ObjectNode
```

Simple POJO Object to JSON in Rest Assured (Serialization) :

Sample simple JSON here used is:

{
"key1":"value1",
"key2":"value2"
}

Passing POJO object directly in Rest Assured body method which will do serialization internally using Jackson or Gson libraries and feed as in application/json format for the request. Also, make note to support this it is needed to include default public constructor in the POJO though we are going to use only parameterized constructor to initialize the values or going to use the setter methods.

```
    @Test
    public void simple_pojo_example() {
   //     SimplePojo simplePojo = new SimplePojo("value1", "value2");
        SimplePojo simplePojo = new SimplePojo();
        simplePojo.setKey1("value1");
        simplePojo.setKey2("value2");

        given().
                body(simplePojo).
        when().
                post("/postSimpleJson").
        then().spec(responseSpecification).
                assertThat().
                body("key1", equalTo(simplePojo.getKey1()),
                        "key2", equalTo(simplePojo.getKey2()));
    }
}
```

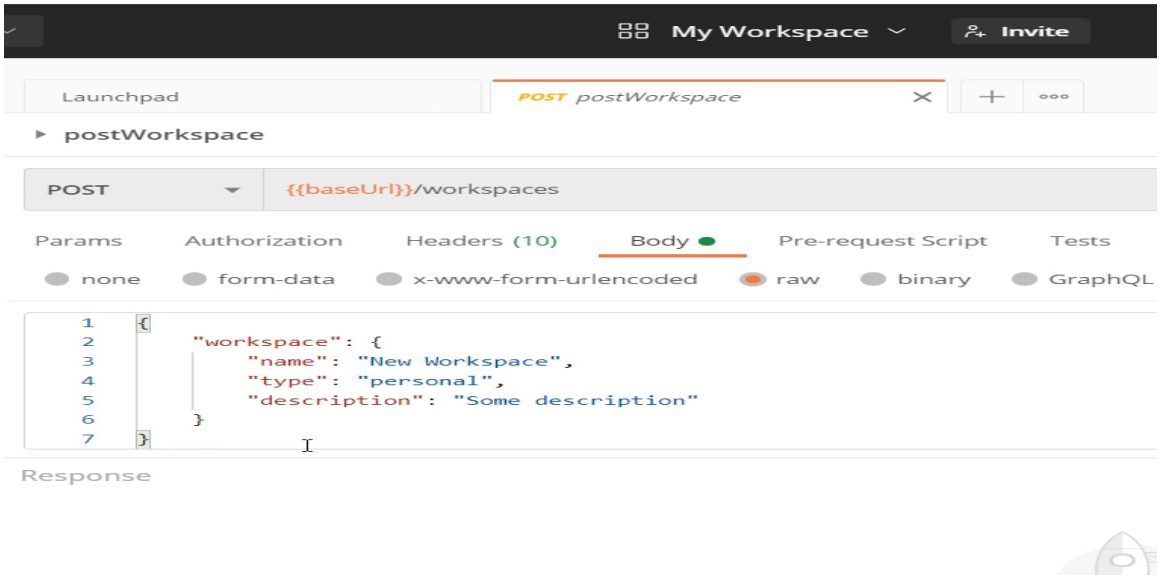
Deserialization of simple POJO:

Below example will receive the response in POJO format and it has to be provided in ***then()*** *condition with the method as where pojo class structure is passed.*

ObjectMapper is an important class in Jackson which here used to convert the POJO objects into JSON values in String formation for comparison.

```
    @Test
    public void simple_pojo_example() throws JsonProcessingException {
        SimplePojo simplePojo = new SimplePojo("value1", "value2");
/*        SimplePojo simplePojo = new SimplePojo();
        simplePojo.setKey1("value1");
        simplePojo.setKey2("value2");*/

        SimplePojo deserializedPojo = given().
                body(simplePojo).
        when().
                post("/postSimpleJson").
        then().spec(responseSpecification).
                extract().
                response().
                as(SimplePojo.class);

        ObjectMapper objectMapper = new ObjectMapper();
        String deserializedPojoStr = objectMapper.writeValueAsString(deserializedPojo);
        String simplePojoStr = objectMapper.writeValueAsString(simplePojo);
        assertThat(objectMapper.readTree(deserializedPojoStr), equalTo(objectMapper.readTree(simplePojoStr)));

    }
}
```

POJO Object for below Payload:
workpsace property root and inside / nested name, type, description



Create POJO with getter , setter for these two nodes:

```java
package com.rest.pojo.workspace;

public class WorkspaceRoot {

    public WorkspaceRoot(){

    }

    1 usage
    public WorkspaceRoot(Workspace workspace) { this.workspace = workspace; }

    public Workspace getWorkspace() { return workspace; }

    public void setWorkspace(Workspace workspace) { this.workspace = workspace; }

    3 usages
    Workspace workspace;
}
```

WorkspacePojoTest.java × Workspace.java × WorkspaceRoot.java × JsonSample.json ×

```java
    7 usages
8   public class Workspace {
        3 usages
9       private String name;
        3 usages
10      private String type;
        3 usages
11      private String description;
12
13      public Workspace(){
14
15      }
        1 usage
16      public Workspace(String name, String type, String description){
17          this.name = name;
18          this.type = type;
19          this.description = description;
20      }
21
22      public String getName() { return name; }
25
26      public void setName(String name) { this.name = name; }
29
30      public String getType() { return type; }
33
34      public void setType(String type) { this.type = type; }
37
38      public String getDescription() { return description; }
41
42      public void setDescription(String description) { this.description = description; }
45  }
46
```

Terminal Services SonarLint Build Dependencies

POJO Serialization and Deserialization in Rest Assured:

```
1 usage
@Test (dataProvider = "workspace")
public void workspace_serialize_deserialize(String name, String type, String description){
    Workspace workspace = new Workspace(name, type, description);
    HashMap<String, String> myHashMap = new HashMap<String, String>();
    WorkspaceRoot workspaceRoot = new WorkspaceRoot(workspace);

    WorkspaceRoot deserializedWorkspaceRoot = given().
            body(workspaceRoot).
    when().
            post( s: "/workspaces").
    then().spec(responseSpecification).
            extract().
            response().
            as(WorkspaceRoot.class);

    assertThat(deserializedWorkspaceRoot.getWorkspace().getName(),
            equalTo(workspaceRoot.getWorkspace().getName()));
}

1 usage
@DataProvider(name = "workspace")
public Object[][] getWorkspace() {
    return new Object[][]{
            {"myWorkspace5", "personal", "description"},
            {"myWorkspace5", "team", "description"}
    };
}
}
```

Serialization

de-Serialization

Deserialization :

If a String property is present extra in the POJO closs, then it will go in the request like below.

```
Body:
{
    "workspace": {
        "id": null,
        "name": "myWorkspace5",
        "type": "personal",
        "description": "description"
    }
}
HTTP/1.1 200 OK
```

*Add Non null annotation from Jackson either at Class level or on separate fields/string members**, then the de-serialized JSON in request will not contain id as it is null.***
Similar to this we have other properties like Non-null (strings), non-default (integer) , non-empty (objects) can be explored in Jackson.

```java
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Workspace {
    private String id;
    private String name;
    private String type;
    private String description;

    public Workspace(){

    }

    public Workspace(String name, String type, String description){
        this.name = name;
        this.type = type;
        this.description = description;
    }

    public String getId() { return id; }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getType() { return type; }

    public void setType(String type) { this.type = type; }
```

```
Body:
{
    "workspace": {
        "name": "myWorkspace5",
        "type": "personal",
        "description": "description"
    }
}
HTTP/1.1 200 OK
```