

API Testing – Training Kit

June 2017



People matter, results count.

Table of Contents

❖ Introduction

- ❖ Pre-requisites for API Testing training
- ❖ Topics covered

❖ API Testing - Introduction

- ❖ What is an API and types of API
- ❖ API Testing – Introduction , Significance
- ❖ Web/Restful Services
- ❖ HTTP Protocol Overview
- ❖ HTTP Request Types – INFO, GET, PUT, POST, DELETE
- ❖ JSON Structure – Syntax – JSONPath
- ❖ HTTP Headers, Payload, Response Codes

Table of Contents... Contd

❖ JUnit

- ❖ What is JUnit
- ❖ JUnit Features
- ❖ JUnit Setup
- ❖ First JUnit Test
- ❖ Framework Overview
- ❖ Annotations
- ❖ Sample JUnit Test

❖ RestAssured

- ❖ Introduction to RestAssured
- ❖ Setup using Maven
- ❖ Sending HTTP requests and Response handling
- ❖ Inspecting fields on the response and Assertion
- ❖ Sample RestAssured Tests

Table of Contents... Contd

❖ Cross-Cutting

- ❖ RAML (Uses etc), Scheme & Schema Validation (What), Error-Warning Propagation, Logging (log4J)

Pre-requisites

- Good hands-on programming using Core Java and OOPS fundamentals
- Basic Testing concepts (Test Case, Test Data, Defect Lifecycle etc..)
- Agile Methodology
- Exposure to Automation

Topics covered

The following modules will be covered during these training sessions in the order listed below

Module	Description	Duration
API Testing Fundamentals	API Types, Web Services, RestFul services, HTTP protocol, HTTP Status/Error codes	8 hours
Junit	Junit Framework, Annotations	2 hours
RestAssured	RestAssured Framework, HTTP request, JSONPath, HTTP Response handling	4 hours
JIRA, Confluence, GIT, CI/CD Pipeline	Overview of various tools and processes used by automation teams – Continuous Integration, Continuous Development	2 hours

API Testing Fundamentals

Definition

Application programming interface (API) is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.

Examples

- Core Windows System Calls (WinAPI)
- BIOS interrupt call
- JNI – Java Native Interfaces
- OpenGL – Exposed as a C library
- Web services – Such as those provided by Facebook's Graph API, etc

API Testing Fundamentals

API vs SDK

API is an interface that allows software programs to interact with each other, whereas a **SDK** is a set of tools that can be used to develop software applications targeting a specific platform.

Examples of SDK include Windows 7 SDK, the Mac OS X SDK, iPhone SDK and of course Oracle Java SDK

Why Test API's ?

- Agile Practices

During certain Agile practices like continuous builds, the amount of time it takes to receive feedback from a GUI regression suite of tests when new code is checked in is too long.

- The Internet of Things

The Internet of Things is an everyday object with embedded functionality that allows it to talk over the web using HTTP or HTTPS to communicate with remote backend services.

API Testing Fundamentals

Web Services

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

API Testing Fundamentals

Web Services

A complete web service is, therefore, any service that :

- Is available over the Internet or private (intranet) networks
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism.

Types of Web Services



API Testing Fundamentals

SOAP (Simple Object Access Protocol) Services

SOAP was originally part of the specification that included the Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). It is used now without WSDL and UDDI. Instead of the discovery process described in the History of the Web Services Specification section below, SOAP messages are hard-coded or generated without the use of a repository. The interaction is illustrated in the figure below.



API Testing Fundamentals

REST

REpresentational State Transfer (REST) is a web standards based architecture and uses HTTP Protocol for data communication. It revolves around resources where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

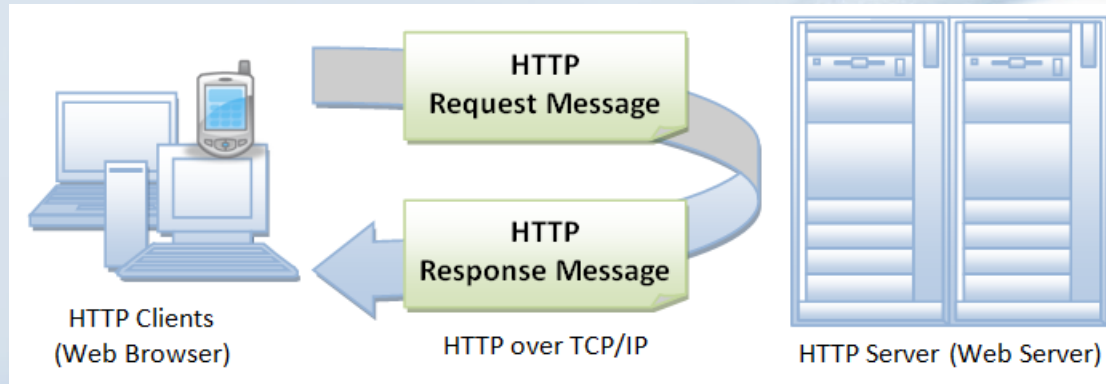
RESTFul Web Services

Web services based on REST Architecture are known as RESTful Web Services. These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI (Uniform Resource Identifier), which is a service that provides resource representation such as JSON and a set of HTTP Methods.

API Testing Fundamentals

HyperText Transfer Protocol (HTTP)

HTTP is an asymmetric request-response client-server protocol. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server

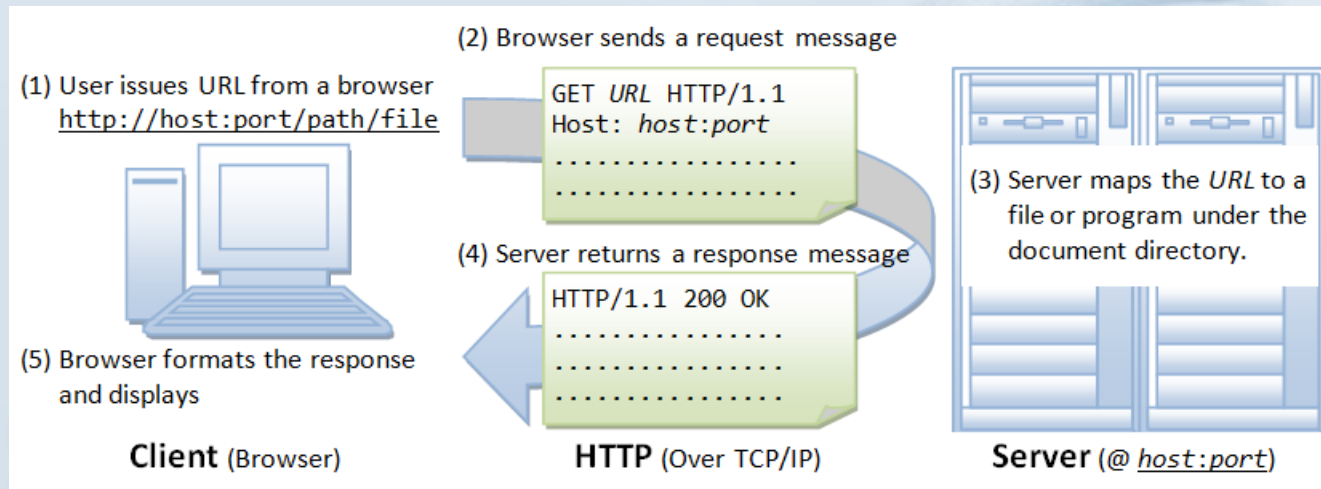


HTTP is a **stateless** protocol. In other words, the current request does not know what has been done in the previous requests.

API Testing Fundamentals

HTTP Communication

Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. `http://www.abc.com/index.html`, the browser turns the URL into a request message and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message.



API Testing Fundamentals

Uniform Resource Identifier(URI)

A URL (Uniform Resource Identifier) is a string of characters used to Identify a resource. A URI identifies a resource either by location, or a name, or both. More often than not, most of us use URIs that defines a location to a resource.

A URI has two specializations known as URL and URN.

URN

A URI identifies a resource by name in a given namespace but not define how the resource maybe obtained. This type of URI is called a URN. You may see URNs used in XML Schema documents to define a namespace, usually using a syntax such as:

```
<xsd:schema    xmlns="http://www.w3.org/2001/XMLSchema"    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:example">
```

Here the targetNamespace use a URN. It defines an identifier to the namespace, but it does not define a location.

```
//example.org/scheme-relative/URI/with/absolute/path/to/resource
```

```
https://example.org/absolute/URI/with/absolute/path/to/resource.txt
```

```
./../resource.txt
```

API Testing Fundamentals

Uniform Resource Locator (URL)

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax:

protocol://hostname:port/path-and-file-name

There are 4 parts in a URL:

Protocol: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.

Hostname: The DNS domain name (e.g., www.abc.com.com) or IP address (e.g., 192.128.1.2) of the server.

Port: The TCP port number that the server is listening for incoming requests from the clients.

Path-and-file-name: The name and location of the requested resource, under the server document base directory.

For example, in the URL <http://www.abc.com/docs/index.html>, the communication protocol is HTTP; the hostname is www.abc.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".

API Testing Fundamentals

HTTP Protocol

Whenever a URL is entered in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

GET /docs/index.html HTTP/1.1

Host: www.abc.com

Accept: image/gif, image/jpeg, */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

(blank line)

When this request message reaches the server, the server can take either one of these actions:

- The server interprets the request received, maps the request into a *file* under the server's document directory, and returns the file requested to the client.
- The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.
- The request cannot be satisfied, the server returns an error message.

API Testing Fundamentals

HTTP Protocol

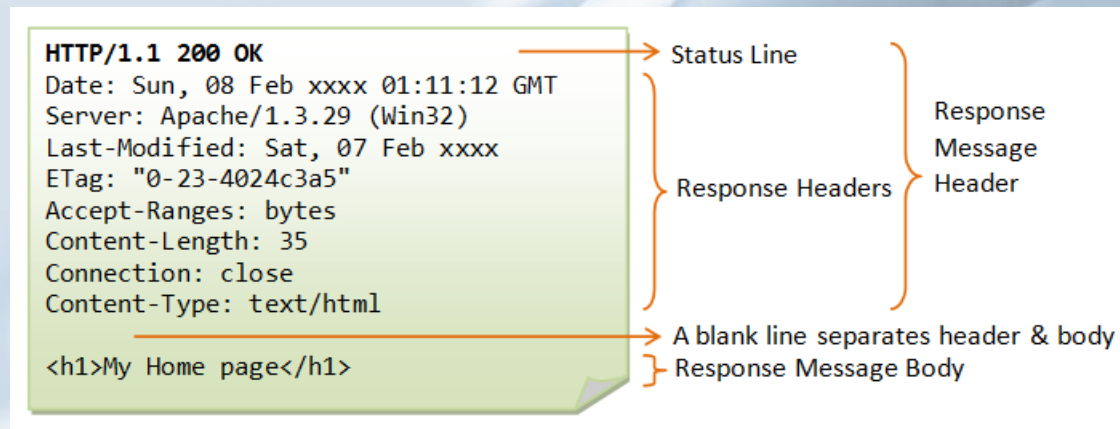
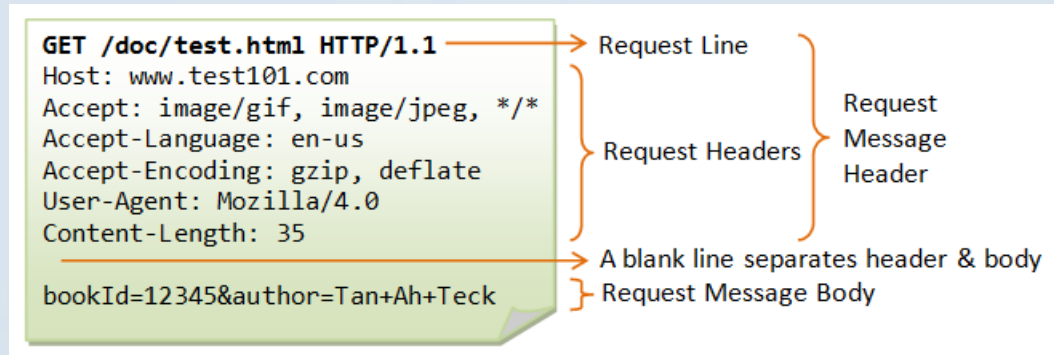
HTTP Response

HTTP/1.1 200 OK
Date: Sun, 07 Jan 2017 09:23:34 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 46
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>Hello World</h1></body></html>

API Testing Fundamentals

HTTP Protocol



API Testing Fundamentals

HTTP Protocol

HTTP Request Types

HTTP protocol defines a set of request types (Verbs). A client can use one of these request methods to send a request message to an HTTP server. The methods are:

GET: A client can use the GET request to get a web resource from the server.

HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.

POST: Used to post data up to the web server.

PUT: Ask the server to store the data.

DELETE: Ask the server to delete the data.

TRACE: Ask the server to return a diagnostic trace of the actions it takes.

OPTIONS: Ask the server to return the list of request methods it supports.

CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.

RestFul clients make use this of these Request Types (Verbs) to interact with the API

API Testing Fundamentals

HTTP Protocol

HTTP Status Codes

The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.

The status code is a 3-digit number:

1xx (Informational): Request received, server is continuing the process.

2xx (Success): The request was successfully received, understood, accepted and serviced.

3xx (Redirection): Further action must be taken in order to complete the request.

4xx (Client Error): The request contains bad syntax or cannot be understood.

5xx (Server Error): The server failed to fulfill an apparently valid request.

API Testing Fundamentals

HTTP Protocol

HTTP Request Headers

These are special instructions/requests that a client can be part of the request – Some of these requests are mandatory and have to be part of the HTTP request for the server to process the request correctly. A typical HTTP request will have one or more of these in these instructions on the request. (**Note** – The server implementation restricts the number of request headers that can be set)

Host: *domain-name*

Accept: *mime-type-1, mime-type-2 (Content Negotiation)*

Accept-Language: *language-1, language-2 (Language Negotiation)*

Accept-Charset: *Charset-1, Charset-2, (Charset Negotiation)*

Accept-Encoding: *encoding-method-1, encoding-method-2,*

Connection: *Close|Keep-Alive*

Referer: *referer-URL*

User-Agent: *browser-type*

Content-Length: *number-of-bytes*

Content-Type: *mime-type*

Authorization

Cookie: *cookie-name-1=cookie-value-1, cookie-name-2=cookie-value-2, ...*

If-Modified-Since: *date*

API Testing Fundamentals

HTTP Protocol

HTTP Response Headers

The response returned by the server can have multiple headers that the client use to process/understand the response. The following are some of the standard responses that can be received by the client.

Access-Control-Allow-Origin	Content-Language	Link	Strict-Transport-Security
Accept-Patch[32]	Content-Length	Location	Trailer
Accept-Ranges	Content-Location	P3P	Transfer-Encoding
Age	Content-MD5	Pragma	Tk
Allow	Content-Range	Proxy-Authenticate	Upgrade
Alt-Svc[33]	Content-Type	Public-Key-Pins[40]	Vary
Cache-Control	Date	Refresh	Via
Connection	ETag	Retry-After	Warning
Content-Disposition[35]	Expires	Server	WWW-Authenticate
Content-Encoding	Last-Modified	Set-Cookie	X-Frame-Options[42]

API Testing Fundamentals

What is JSON

JSON is a simple, text-based way to store and transmit structured data. By using a simple syntax, you can easily store anything from a single number through to strings, arrays, and objects using nothing but a string of plain text. You can also nest arrays and objects, allowing you to create complex data structures..

Advantages of JSon

- It's compact
- Easy to comprehend
- It maps very easily onto the data structures used by most programming languages (numbers, strings, booleans, nulls, arrays and associative arrays)
- Nearly all programming languages contain functions or libraries that can read and write JSON structures

API Testing Fundamentals

Sample Json Object

```
{
  "orderId": 12345,
  "shopperName": "Srinivas Ravuri",
  "shopperEmail": "srin.ravuri@capgemini.com",
  "contents": [
    {
      "productId": 34,
      "productName": "TV",
      "quantity": 1
    },
    {
      "productId": 56,
      "productName": "Samsung",
      "quantity": 3
    }
  ],
  "orderCompleted": true
}
```

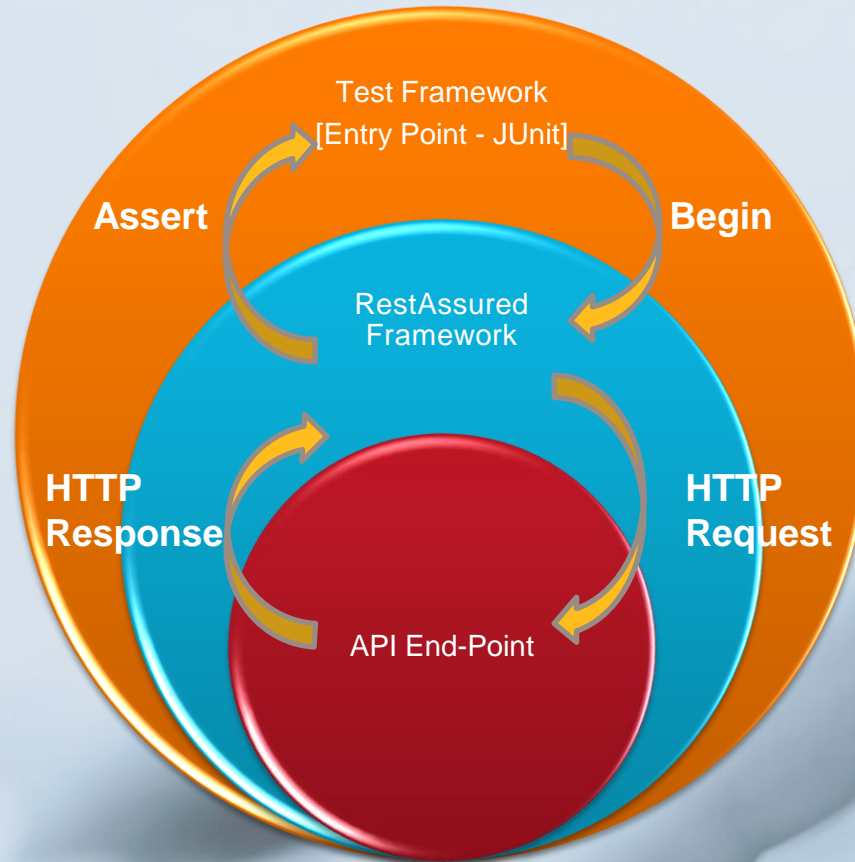

API Testing Fundamentals

JSon Rules

- A JSON string contains either an array of values, or an object (an associative array of name/value pairs).
- An array is surrounded by square brackets, [and], and contains a comma-separated list of values.
- An object is surrounded by curly brackets, { and }, and contains a comma-separated list of name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon (:), followed by the field value.
- A value in an array or object can be:
 - A number (integer or floating point)
 - A string (in double quotes)
 - A boolean (true or false)
 - Another array (surrounded by square brackets, [and])
 - Another object (surrounded by curly brackets, { and })
 - The value null

Note: RestFul services use JSon as payload over an HTTP request

API Testing Model



JUnit

What is JUnit

- JUnit is a Unit testing framework for Java Programming Language.
- The Framework is available in the form of an “JAR” (Java Archive) that must be included during compile time.
- It is open source.

JUnit Features

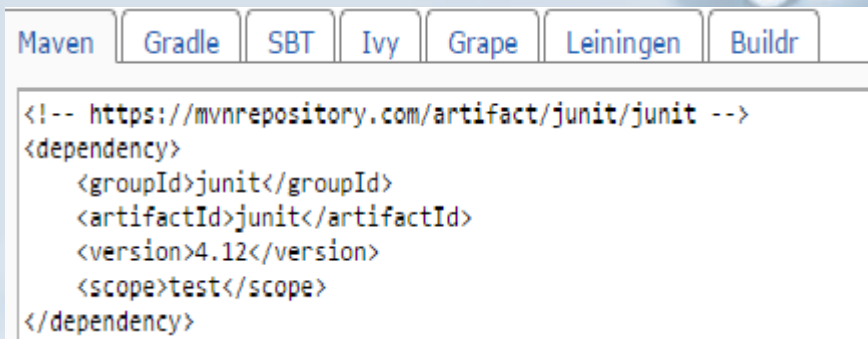
- Open Source framework for creating and executing tests
- Annotations for Identifying Test Methods.
- Assertions – Compare Expected vs Actual Results.
- Test Grouping into test suites

JUnit

JUnit Setup

JUnit can be setup in one of the following ways :

- A) Download the latest jar and add it to CLASSPATH
 - SET CLASSPATH=%CLASSPATH%;junit4.12.jar; (**Windows**)
 - CLASSPATH=\$CLASSPATH:junit4.12.jar: (Unix/Linux/Solaris)
- B) Add JUnit dependency in your build setup (Maven, Gradle etc..)

A screenshot of a build tool's dependency configuration interface. At the top, there are tabs for different build tools: Maven, Gradle, SBT, Ivy, Grape, Leiningen, and Buildr. The 'Maven' tab is selected. Below the tabs, there is a text area containing XML code for a Maven dependency. The code is as follows:

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

JUnit

First JUnit Test

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestJUnit
{
    @Test
    public void testAdd()
    {
        String str= "This is a test";
        assertEquals("This is a test",str);
    }
}
```

From Eclipse IDE, you can execute the above class “As a JUNIT” test or to run it from command line – Use the following

```
java -cp junit4.12.jar org.junit.runner.JUnitCore TestJUnit
```

JUnit

JUnit Test Framework

Fixtures is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable. It includes:

- * *setUp()* method, which runs before every test invocation.
- * *tearDown()* method, which runs after every test method.

Test Suites

A test suite bundles a few unit test cases and runs them together. In JUnit, both *@RunWith* and *@Suite* annotation are used to run the suite test.

Test Runners

Test Runner is used for executing the test cases (This features is not extensively used) - JUnit provides JUnitCore class that can "Introspect" (Reflection) a test class and execute all annotated methods (*@Test*, *@BeforeTest*, *@AfterTest* etc...)

JUnit Classes

The framework provides Classes that are used for writing test cases. Examples include :

- ✓ TestCase
- ✓ TestResult
- ✓ Assert

Annotations

Annotations are like meta-tags that you can add to your code, and apply them to methods or in class. (Example - *@Test*, *@Before* etc.)

JUnit

Annotations

Annotations are like meta-tags that you can add to your code and apply them to methods or in class. These annotations in JUnit provide the following information about test methods :

- Which methods are going to run before and after test methods,
- Which methods run before and after all the methods and
- Which method/class will be ignored during execution.

Annotation	Description
@Test	The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.
@Before	Several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before each Test method.
@After	If you allocate external resources in a Before method, you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method.
@BeforeClass	Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class.
@AfterClass	This will perform the method after all tests have finished. This can be used to perform clean-up activities.
@Ignore	The Ignore annotation is used to ignore the test and that test will not be executed.

JUnit

```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Ignore;
import org.junit.Test;
public class JunitAnnotation
{
    //execute before class
    @BeforeClass
    public static void beforeClass()
    {
        System.out.println("Connecting to database....");
        dbHandler = dbHandler.connectTo(""); // Connect to a database...
    }

    //execute after class
    @AfterClass
    public static void afterClass()
    {
        System.out.println("Drop db connection and cleaning up");
        dbHandler.disconnect(); // Disconnect
    }

    //execute before test
    @Before
    public void before()
    {
        System.out.println("Read the data file");
        dataFile.read(); // Load the data to use...
    }
}
```

JUnit

```
//execute after test
@After
public void after()
{
    System.out.println("Load the next set of data");
    dataFile.useNext(); // Load the next set of data for a test...
}

//test case
@Test
public void test()
{
    System.out.println("Actual test method - This will be invoked only ones");
}

//test case ignore and will not execute
@Ignore
public void ignoreTest()
{
    System.out.println("This test is being ignored in the current run"); // This method is never executed.
}
}
```

Logging (log4J)

Primarily used for debugging, tracing the sequence of execution and monitoring.

Logging frameworks - Log4j, slf4j, Java Logging API (Part of JRE) etc.

Advantages over traditional `system.out.println()`

- Configuration driven - Provide better way of "controlled" logging. Logging level can be defined.
- Support for Appenders - These are components that are responsible for printing logging messages to different destinations (Files, Consoles, Sockets, NT event logs etc)
- Define layouts - Simple text, HTML, Custom pattern layout, Date/time stamp layout.

Features

- Configuration stored in `log4j.properties` file.
- Content can be logged based on the required granularity (Logging levels)

Logging (log4J)

Logging Levels

Level	Description	Method To Invoke
DEBUG	Designates fine-grained informational events that are most useful to debug an application.	logger.debug
ERROR	Designates error events that might still allow the application to continue running.	logger.error
FATAL	Designates very severe error events that will presumably lead the application to abort.	logger.fatal
INFO	Designates informational messages that highlight the progress of the application at coarse-grained level.	logger.info
TRACE	Designates finer-grained informational events than the DEBUG.	logger.trace
WARN	Designates potentially harmful situations.	logger.warn
ALL	All levels including custom levels.	
OFF	The highest possible rank and is intended to turn off logging.	

ALL < DEBUG < INFO < TRACE < WARN < ERROR < FATAL < OFF

Logging (log4j)

log4j.properties – Logging onto console

```
# Root logger option
log4j.rootLogger=INFO, stdout
```

```
# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

log4j.properties – Logging intoFile

```
log4j.rootLogger=INFO, file
# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
#Redirect to Tomcat logs folder
#log4j.appender.file.File=${catalina.home}/logs/logging.log
log4j.appender.file.File=C:\\logigng.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```


RestAssured

What is RestAssured

- REST Assured is a Java DSL for simplifying testing of REST based services built on top of HTTP Builder. It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests.

RestAssured Features

- Supports handling of interacting with request and response headers
- Strong library support to handle XML/JSON responses.
- Open source
- Framework ideally suited for testing of RestFul services.
- Supports Given, When, Then constructs to increase java code readability (Not highly recommended)

RestAssured

RestAssured Setup

- The recommended way to setup RestAssured is to include the Maven dependency in your project pom file – This ensures that all dependencies are automatically downloaded and are available in your build path.
- Include the following in your project pom.xml
- The latest version of RestAssured is 2.9.0

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/com.jayway.restassured/rest-assured -->
<dependency>
  <groupId>com.jayway.restassured</groupId>
  <artifactId>rest-assured</artifactId>
  <version>2.9.0</version>
  <scope>test</scope>
</dependency>
```

RestAssured

given()/with()

- The HTTP request must be constructed first before we can send it to an API. RestAssured provides a method "given", with() that let's us create a HTTP request. They returns an object of type "RequestSpecification".
- *Note - By default, RestAssured generates an HTTP GET request with no additional parameters on the headers. In the below examples, given/with/when can be used interchangeably.*
- RequestSpecification provides various methods that will let us modify the HTTP request.

Example: `given().body("{\"message\": \"hello world\"})`

In the example above, we are changing the payload (Body).

- To set cookies on the HTTP request, we can invoke the cookies method.

Example: `given().cookies("username", "srinir", "token", "77#123")`

- **param() method let's you declare parameters**

Example: `with().param("name1", "value1")`

RestAssured

when()/then()/expect()

- The HTTP request that is constructed by any of the above 2 means is sent using the when()/then() method. they do two distinct operations:
 - Provides the CRUD operation that needs to be performed via get(), put(), post(), delete() methods (And others)
 - Allows you to specify how the expected response must look like in order for a test to pass.
- ResponseSpecification provides various methods that will let us inspect the HTTP response.
Example: `when().get("/something").then().header("Content-Length", Integer::parseInt, lessThan(600));`

`expect().content(containsString("summary-balance")).when().get("/statement");`

*Note – It is not required to follow the **given.when.then** convention.*

More examples to be covered as part of hands-on exercises.

RestAssured

Extracting StatusCode from Response

```
@Test
public void getStatusCodeTest()
{
    System.out.println("Connecting to the Service...");
    try
    {

        Response r =
            given().when().get("http://api.openweathermap.org/data/2.5/weather?id=1255816&appid=42b296c9ecf85b63566d9f89e24d4a7c").then().extract().response();
        System.out.println("Status Code : " + r.getStatusCode());
    }
    catch(Exception ex_)
    {
        // Do nothing;
    }

}
```


RestAssured

Inspecting Header Elements on Response

```
@Test
public void getAllHeaders()
{
    System.out.println("Connecting to the Service...");
    try
    {

        Response r =
            given().when().get("http://api.openweathermap.org/data/2.5/weather?id=1255816&appid=42b296c9ecf85b63566d9f89e24d4a7c").then(
            ).extract().response();
        System.out.println("Status Code : " + r.getStatusCode());
        System.out.println("Response Header");

        Headers allOptions = r.getHeaders();
        List allHeaders = allOptions.asList();
        for(int a = 0; a < allHeaders.size(); a++)
        {
            System.out.println(allHeaders.get(a));
        }
    }
    catch(Exception ex_)
    {
        // Do nothing;
    }
}
```

RestAssured

Setting Parameters on the Request and other validations on Response

```
@Test
public void parameterTest()
{
    // http://api.openweathermap.org/data/2.5/weather?id=&appid=42b296c9ecf85b63566d9f89e24d4a7c
    System.out.println("Parameter Test...");

    given().parameters("id", "1255816", "appid", "42b296c9ecf85b63566d9f89e24d4a7c").when().
    get("http://api.openweathermap.org/data/2.5/weather").then().assertThat().statusCode(400);
}
```

Cookie Validation on Response

```
get("/someURI").then().assertThat().cookie("cookieName", "cookieValue"). ..
```

Status Line Validation on Response

```
get("/someURI").then().assertThat().statusCode(400). ..
```

Header Field Validation on Response

```
get("/someURI").then().assertThat().header("headerName", "headerValue"). ..
```

Extracting the Response Time

```
long timeInMs = get("/someURI").time();
```

RAML

- RESTful API Modeling Language
- Provides necessary information to describe RESTful API's
- Information include Resource Names, Supported actions (GET/PUT/POST/DELETE etc), Traits, Security Schemas
- QA Teams can refer the RAML spec and understand “How” to invoke an API – The parameters that are supported against a given resource, operations and “What” to expect in a response

RestAssured

Json Schema Validation with RestAssured

```
import org.junit.Test;
import static io.restassured.module.json.JsonSchemaValidator.matchesJsonSchemaInClasspath;
import static org.hamcrest.MatcherAssert.assertThat;

public class JsonSchemaValidatorWithoutRestAssuredTest {

    @Test public void validates_schema_in_classpath() {
        // Given
        String json = ... // Json response

        // Then
        assertThat(json, matchesJsonSchemaInClasspath("city-weather-schema.json"));
    }
}
```



Thank You