**What is a web service?**

* Web services are the client-server applications for communication purpose. i.e. enables communication between applications over the web.
* Web services are a platform independent & language-independent way of communication.

Diagram, schematic

Description automatically generated

* A web service is a set of standards protocols/format that allow data to be exchanged between different applications or systems.
* Web services can be used by software programs written in a variety of programming languages and running on a variety of platforms to exchange data via computer networks such as the Internet.
* It provides service available over the web(WWW)
* using web services two different applications can talk to each other and exchange data/information
* All web services are APIs.

**What is an API?**

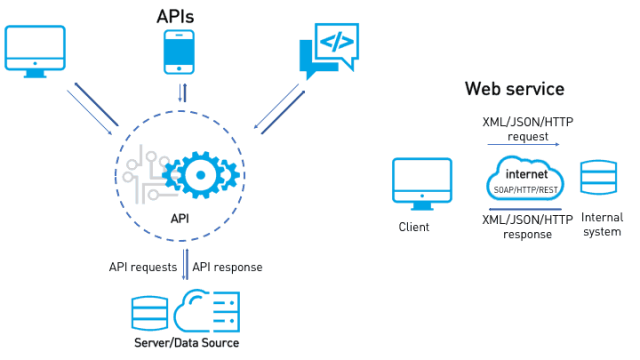
* API stands for Application Programming Interface. It is a software interface that allows two applications to interact with each other without any user intervention.
* APIs provides product or service to communicate with other products and services without having to know how they’re implemented.

**Difference between Web Service and API ?**

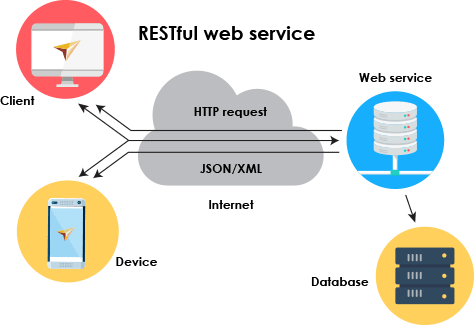
1. Webservices always need network for its operations, but API may or may not need network for its operations
2. Webservice can be used by following communication : Soap, Rest, XML RPC but API is used for any style of communication & other ways like cURL calls.

( cURL, which stands for client URL, is a command line tool that developers use to transfer data to and from a server. At the most fundamental, cURL lets you talk to a server by specifying the location (in the form of a URL) and the data you want to send )

1. API can also be exposed in number of ways like JAR, DLL, XML over Http, Json over Http etc.
2. Web service supports only HTTP protocol whereas API supports HTTP/HTTPS protocol.
3. Web service supports XML while API supports XML and JSON.
4. API supports traditional CRUD (Create Read Update Delete) actions as it works with HTTP verbs GET, PUT, POST, and DELETE.
5. All Webservice can be consider as API i.e. (SOAP and REST are the APIs), but all API are not Webservices.



**Concept behind web services:**



1. **Server (service provider)**

A web service provider develops/implements the web service and makes it available over the internet (web)

1. **Client (Service consumer)**

The client is the person or software who use the APIs. A browser can be a client because it calls the APIs and return the data to render the information’s on the screen.

1. **Medium**

HTTP or Internet

1. **Format**

XML or JSON

**Example**: While speaking to your friend over telephone, Medium is the **Phone** and Format is the common **language** (e.g. English) that both of you can understand.

**Types of Web Services?**

There are two types of web services:

1. SOAP Web Services.
2. REST Web Services.

Diagram

Description automatically generated

**SOAP**:

* SOAP is a protocol which was designed before REST came into the picture. The main idea behind creating SOAP was to ensure that programs built on different platforms and programming languages could securely exchange data.
* Every programming language can understand the XML markup language. Hence, XML was used as the underlying medium for data exchange. Hence we call it as XML based protocol for accessing web services.
* It is both platform and language independent.
* A web service that complies to the SOAP Web Services Specifications is a SOAP Web Service.

Soap specifications / standards & Who defines and dictates these standards ?

* **Basic** standards are **SOAP** (All information/message exchange happens over a common format: XML), **WSDL**(XML messages have a defined structure: SOAP MESSAGE) & **UDDI**
* **Extended** standards are WS-Security
* W3C will defines these standards (World Wide Web Consortium)
* An international community that develops open standards for world wide web.

SOAP stands for **Simple Object Access Protocol**

**Medium**: HTTP (POST)

**Format** : XML based protocol for accessing web services.

Diagram

Description automatically generated

**Advantages of using SOAP:**

* **Security** — SOAP defines its own security known as ws security.
* **Language and platform-independent** SOAP web services can write in any language and execute on any platform.
* SOAP has transactions with ACID properties.

**Disadvantages of using SOAP**

* **Slow**- SOAP uses XML format and that need to parse to read. And SOAP defines many strict standards must be followed. consumes more bandwidth.
* **WSDL dependent**- SOAP uses WSDL(Web Service Description Language) and does not have any other mechanism to discover the services.

**REST:**

REST stands for Representational State Transfer

Medium : HTTP (POST, GET, PUT, DELETE,...)

Format: XML/JSON/TEXT/HTML...

REST stands for Representational State Transfer, just like regular websites, REST APIs use HTTP for communication. The data can be sent or received in the form of JSON or XML.

A web service that communicates / exchanges information between 2 applications using REST architecture/principles is called a RESTful Web Service.

**REST** does not use any standard or predefined structure to validate data. It sends the request message and gives us the response. It uses HTTP methods to send or receive the data.

HTTP methods are :

1. **POST** : POST Request to create a new user.
2. **GET** : GET request to get the existing users.
3. **PUT** : PUT request to update the details of existing user.
4. **DELETE** : DELETE request to delete an existing user.

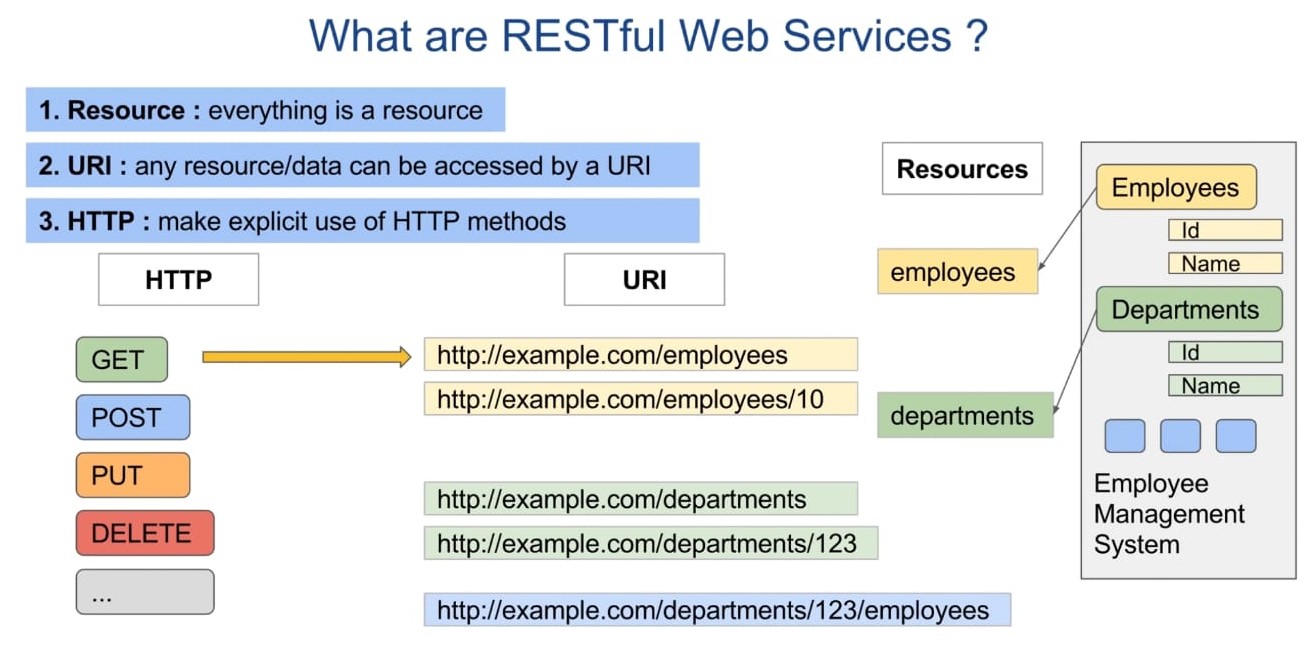
**What rules should a web service follow to become a RESTful Web Service ?**

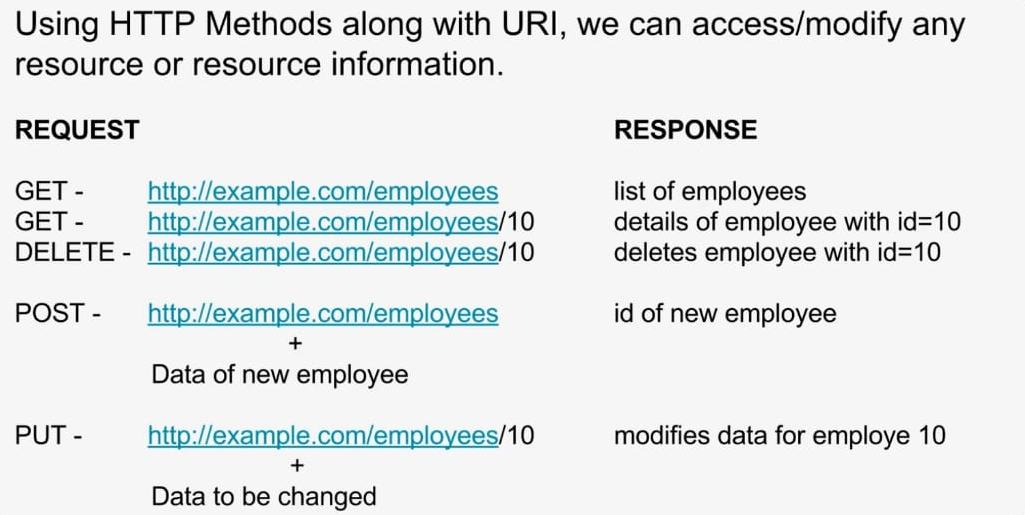
* REST defines a set of principles to be followed while designing a service for communication / data exchange between 2 applications.
* When these principles are applied while designing web services (for client - server interactions) we get : RESTful Web Services.

**What are the principles/constraints of REST architecture?**

It follows these **6** principles :

1. **Client-Server** : This concept explains that the client and the server should be separate from each other and allowed to evolve individually.
2. **Stateless** : REST APIs are stateless, meaning that it does not save history of any request or response. Once the request is sent by the client it closes the connection and establishes the new connection to give the response.
3. **Cache** : Because a stateless API can increase request overhead by handling large loads of incoming and outbound calls, because for each request and response it creates a new connection, a REST API should be designed to encourage the storage of cacheable data.
4. **Uniform Interface** :
   1. Resource : everything is a resource
   2. URI : any resource/data can be accessed by a URI
   3. HTTP : make explicit use of HTTP methods





1. **Layered System** : REST APIs have different layers of their architecture working together to build a hierarchy that helps create a more scalable and modular application.
2. **Code on Demand** : Code on Demand allows for code or applets to be transmitted via the API for use within the application .

**SOAP V/S REST :**

**Data Transfer:**

**SOAP** uses **WSDL** (**W**eb **S**ervice **D**escriptive **L**anguage) for data validation. It validates the data with WSDL then proceed for the data transfer else it gives the error message and terminates the data transfer process.

**WSDL** :

* Is an XML based interface that is used to describe the functionalities of the Web Services
* As its name indicates it is a description or a specific standard to validate the SOAP data. WSDL validates the data with .xsd file which have predefined structured format. A SOAP request has these elements :

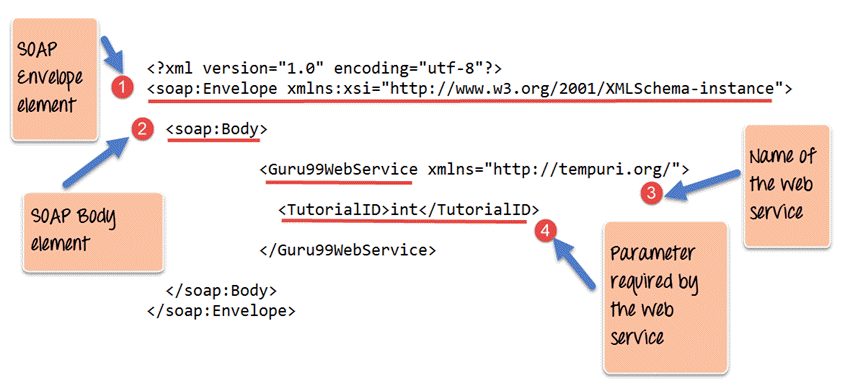
1. **Envelope** : Envelope is the root element of a Soap Message.

This is the basic unit of the XML document which contains other units like Header and Body.

1. **Header** : Header element provides information about the message itself.

Header might include authentication, complex types, routing information etc.

1. **Body** : Body contains the actual data of the request meant to be sent to the server.
2. **Fault** : An **optional** Fault element that provides information about errors that occur while processing the message. Refer the image for SOAP request.

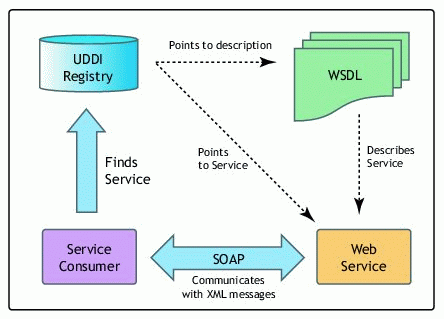


Example:

<https://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl>

**UDDI:**

* UDDI stands for Universal Description, Discovery, and Integration
* A web service provider publishes his web service (through wsdl) on an online directory from where consumers can query and search the web services. This online registry/directory is called UDDI
* Is an XML based standard for publishing and finding web services.



**Secure Transfer** :

* REST API is the stateless protocol that does not maintain the record of any transaction , from the security point of view it is not good to use REST for confidential data sharing.
* SOAP provides the secure connection to the server for sending request and receiving response.

**Data Manipulation :**

**SOAP** follows the **ACID** property that ensures the data should not be alter in between the transaction.

**REST** does not follow any of the standard to maintain data manipulation.

**ACID** property stands for **Atomicity , Consistency , Isolation , Durability**.

* **Atomicity** : Atomicity guarantees that each transaction is treated as a single “unit”, which either succeeds completely, or fails completely.
* **Consistency** :Consistency ensures that the data should be validated with the predefined rules.
* **Isolation** : Isolation ensures that concurrent execution of transactions does not affect by other transactions .
* **Durability** : Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure.

**Processing Time :**

REST process the data faster than the SOAP API, as it uses JSON data format whereas SOAP uses XML format for data.

|  |  |
| --- | --- |
| Difference between URL & URI | |
| URL | URI |
| URL stands for Uniform Resource Locator. | URI stands for Uniform Resource Identifier. |
| The main aim is to get the location or address of a resource | The main aim of URI is to find a resource and differentiate it from other resources using either name or location. |
| URL is used to locate only web pages | Used in HTML, XML, and other files XSLT (Extensible Stylesheet Language Transformations) and more. |
| All URLs can be URIs | Not all URIs are URLs since a URI can be a name instead of a locator. |

**HTTP request methods:**

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.

**GET :** The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

**HEAD :** The HEAD method asks for a response identical to a GET request, but without the response body.

**POST :** The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

**PUT :** The PUT method replaces all current representations of the target resource with the request payload.

**DELETE :** The DELETE method deletes the specified resource.

**OPTIONS :** The OPTIONS method describes the communication options for the target resource.

**PATCH :** The PATCH method applies partial modifications to a resource.

**HTTP response status codes**

HTTP response status codes indicate whether a specific HTTP request has been successfully completed.

NOTE : <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Responses are grouped in five classes:

1. **Informational responses (100–199)**
   1. 100 Continue
   2. 101 Switching Protocols
   3. 102 Processing
2. **Successful responses (200–299)**
   1. 200 OK
   2. 201 Created
   3. 202 Accepted
   4. 204 Deleted
3. **Redirection messages (300–399)**
   1. 300 Multiple Choices
   2. 301 Moved Permanently
   3. 302 Found
   4. 303 See Other
   5. 304 Not Modified
   6. 305 Use Proxy
4. **Client error responses (400–499)**
   1. 400 Bad Request
   2. 401 Unauthorized
   3. 402 Payment Required
   4. 403 Forbidden
   5. 404 Not Found
   6. 408 Request Timeout
   7. 415 Unsupported Media Type
   8. 423 Locked
   9. 429 Too Many Requests
   10. 431 Request Header Fields Too Large
   11. 444 Connection Closed Without Response
   12. 499 Client Closed Request
5. **Server error responses (500–599)**
   1. 500 Internal Server Error
   2. 501 Not Implemented
   3. 502 Bad Gateway
   4. 503 Service Unavailable
   5. 504 Gateway Timeout
   6. 505 HTTP Version Not Supported
   7. 507 Insufficient Storage
   8. 511 Network Authentication Required
   9. 599 Network Connect Timeout Error

**What is REST Assured ?**

* REST Assured is a java library for testing RESTful web services
* It is used to invoke REST web services and check response
* Can be used to test XML as well as JSON based web services
* It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests
* Can be integrated with testing frameworks like JUnit, TestNG, BDD etc.

**Prerequisites**

* Java
* IDE (Eclipse, IntelliJ, etc.)
* Maven
* TestNG

**Create project for REST Assured**

Step 1: Open Eclipse

Step 2: Create a maven project

Step 3: Add dependencies in pom.xml

Step 4: Save project

Step 5: Check libraries added

Step 6: Add TestNG plugin in eclipse

**Create 1st Test**

Step 1: Create a class

Step 2: Create a function and annotate with @Test (TestNG)

Step 3: Run a GET request

Step 4: Store response and print response data

Step 5: Add assertions

Step 6: Run and verify

Code: GET

import io.restassured. RestAssured;

import io.restassured.response.Response;

public class TestsExamples

{

@Test

public void test\_1()

{

Response response = RestAssured.get("https://reqres.in/api/users?page=2");

System.out.println(response.getStatusCode());

System.out.println(response.getTime());

System.out.println(response.getBody().asString());

System.out.println(response.getStatusLine());

System.out.println(response.getHeader("content-type"));

int statusCode = response.getStatusCode();

Assert.assertEquals(statusCode, 200);

}

}

OR

NOTE : <https://jsonpathfinder.com/>

import io.restassured. RestAssured;

import **static** io.restassured.RestAssured.\*;

import static org.hamcrest. Matchers.\*

public class TestsExamples

{

@Test

public void test\_2()

{

baseURI= "https://reqres.in/api";

given()

.get("/users?page=2")

.then()

.statusCode(200)

.body("data.id[1]", equalTo(8))

.body("data.first\_name", hasltems("George", "Rachel"));

.log().all();

}

}

POST:

import io.restassured. RestAssured;

import io.restassured.response.Response;

public class TestsExamples

{

@Test

public void test\_post()

{

Map<String, Object> map = new HashMap<String, Object>();

map.put("Name", "Raghav");

map.put("job", "Teacher");

System.out.println(map);

JSONObject request = new JSONObject(map);

System.out.println(request.toJSONString());

baseURI= "https://reqres.in/api";

given()

.body(request.toJSONString())

.when()

.post(“/users”)

.then()

.statusCode(**201**);

}

}

OR

import io.restassured. RestAssured;

import io.restassured.response.Response;

public class TestsExamples

{

@Test

public void test\_post()

{

JSONObject request = new JSONObject(map);

request.put("Name", "Raghav");

request.put("job", "Teacher");

System.out.println(request.toJSONString());

baseURI= "https://reqres.in/api";

given()

.header("Content-Type", "application/json")

//or

.contentType(ContentType.JSON) // request

.accept(ContentType.JSON) //response

.body(request.toJSONString())

.when()

.post(“/users”)

.then()

.statusCode(201)

.log().all();

}

}

PUT:

import io.restassured. RestAssured;

import io.restassured.response.Response;

public class TestsExamples

{

@Test

public void test\_put()

{

JSONObject request = new JSONObject(map);

request.put("Name", "Raghav");

request.put("job", "Teacher");

System.out.println(request.toJSONString());

baseURI= "https://reqres.in/api";

given()

.header("Content-Type", "application/json")

//or

.contentType(ContentType.JSON) // request

.accept(ContentType.JSON) //response

.body(request.toJSONString())

.when()

.put(“/users/2”)

.then()

.statusCode(200)

.log().all();

}

}

PATCH:

**import** io.restassured. RestAssured;

**import** io.restassured.response.Response;

**public** **class** TestsExamples

{

@Test

**public** **void** test\_patch()

{

JSONObject request = **new** JSONObject(map);

request.put("Name", "Raghav");

request.put("job", "Teacher");

System.out.println(request.toJSONString());

baseURI= "https://reqres.in/api";

given()

.header("Content-Type", "application/json")

//or

.contentType(ContentType.JSON) // request

.accept(ContentType.JSON) //response

.body(request.toJSONString())

.when()

.patch(“/users/2”)

.then()

.statusCode(200)

.log().all();

}

}

DELETE:

**import** io.restassured.RestAssured;

**import** io.restassured.response.Response;

**public** **class** TestsExamples

{

@Test

**public** **void** test\_delete()

{

baseURI= "https://reqres.in/api";

when()

.delete("/users/2")

.then()

.statusCode(204)

.log().all();

}

}

Different types of json libraries :

1. Gson
2. Jackson
3. Json in java
4. Json simple

**validate JSON Schema**

Step 1: Create JSON Schema

Step 2: Add JSON Schema file in classpath i.e.

>target > classes> sample.json

Step 3: Add Maven dependency for JSON Schema Validator

* + - **JSON schema validator 4.3.1**

Step 4: Create a new function to validate json response against schema

Step 5: Run & Validate

**import** **static** io.restassurea.module.jsv.Jsonscnemavallaator.matcnessonscnemainclasspatn;

**import** **static** org.hamcrest.Matchers.equalTo;

**import** **static** org.hamcrest.Matchers.hasItems;

**import** org.testng.annotations.Test;

**public** **class** JSONSchemaValidator

{

@Test

**public** **void** testGet()

{

baseURI= "https://reqres.in/api";

given().

then().

get("/users?page=2").

assertThat().body(matchesJsor Schema In Classpath("schema.json")).

statusCode(200);

}

}

**SOAP API with XML Response:**

Step 1: Create a class

Step 2: Create a function and annotate with @Test (TestNG)

Step 3: Add baseURI

Step 4: Provide headers and body

given().

contentType("text/xml").

header("Accept", "text/xml").

body(requestBody)

**To get request body from a file**

Step 1: Create a file with xml extension

Step 2: Copy the request body and save file

Step 3: Get the file in the code

FileInputStream fileInputStream = new FileInputStream(new File("./SOAPRequest/add.xml")); String requestBody = IOUtils.toString(fileInputStream, "UTF-8");

**Note**: IOUtils comes from apache.commons

If required get the pom dependency

Code:

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** **static** io.restassured. RestAssured.baseURI;

**public** **class** SoapXMLRequest

{

@Test

**public** **void** validateSoapXML () **throws** IOException

{

File file = **new** File("./SoapRequest/Add.xml");

**if**(file.exists())

System.out.println(">> File Exists");

FileInputStream fileInputStream = **new** FileInputStream(file);

String requestBody = IOUtils.toString(fileInputStream, "UTF-8");

baseURI = "http://www.dneonline.com";

given().

contentType("text/xml").

accept(ContentType.XML).

body(requestBody).

when().

post("/calculator.asmx").

then().

statusCode(200);

}

}

**validate XML Schema**

Step 1: Create or Get XML Schema (XSD)

Step 2: Add XSD to classpath in the project -src/main/resources

Step 3: Add a new class

Step 4: Create a function > Annotate with @Test (TestNG)

Step 5: Add code to run a API request with REST assured

Step 6: Add code to validate XML response against XML Schema (XSD)

assertThat().body(matchesXsdInClasspath("calculator.xsd"));

import static io.restassured.matcher.RestAssured Matchers.matchesXsdInClasspath;

Code:

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** **static** io.restassured. RestAssured.baseURI;

**import** **static** io.restassured.matcher.RestAssured Matchers.matchesXsdInClasspath;

**public** **class** SoapXMLRequest

{

@Test

**public** **void** validateSoapXML () **throws** IOException

{

File file = **new** File("./SoapRequest/Add.xml");

**if**(file.exists())

System.out.println(">> File Exists");

FileInputStream fileInputStream = **new** FileInputStream(file);

String requestBody = IOUtils.toString(fileInputStream, "UTF-8");

baseURI = "http://www.dneonline.com";

given().

contentType("text/xml").

accept(ContentType.XML).

body(requestBody).

when().

post("/calculator.asmx").

then().

statusCode(200).

and().

assertThat().body(matchesXsdInClasspath("Calculator.xsd"));

}

}