For any web application, the API services work as the **building block**. Therefore it becomes very important for you, as a tester, to ensure that the **web services are bug-free** and facilitate the **exchange of information securely**.

**What is Client Server Architecture?**

Client server architecture in a simple sense can be stated as a **consumer-producer model where the client acts as the consumer i.e. the service requestor and the server is the producer, *i.e.* the service provider**. Let us see how they are described in the computing sense.

***What is a Client?***

A client in computing is a system or a program that connects with a remote system or software to fetch information. **The client makes a request** to the server and is responded with information. There may be three types of clients - thick, thin or hybrid client.

* ***A thick or fat client performs the operation itself without the need for a server****.* ***A personal computer*** *is a fine example of a thick client as the majority of its operations are independent of an external server.*
* ***A thin client uses the host computer's resources, where the thin client presents the data processed by an application server. A web browser is an example of a thin client.***
* ***A hybrid client is a combination of a thick and a thin client****. Just like the thick client it processes data internally but relies on the application server for persistent data.* ***Any online gaming run*** *on a system can be an example of a hybrid client.*

***What is a Server?***

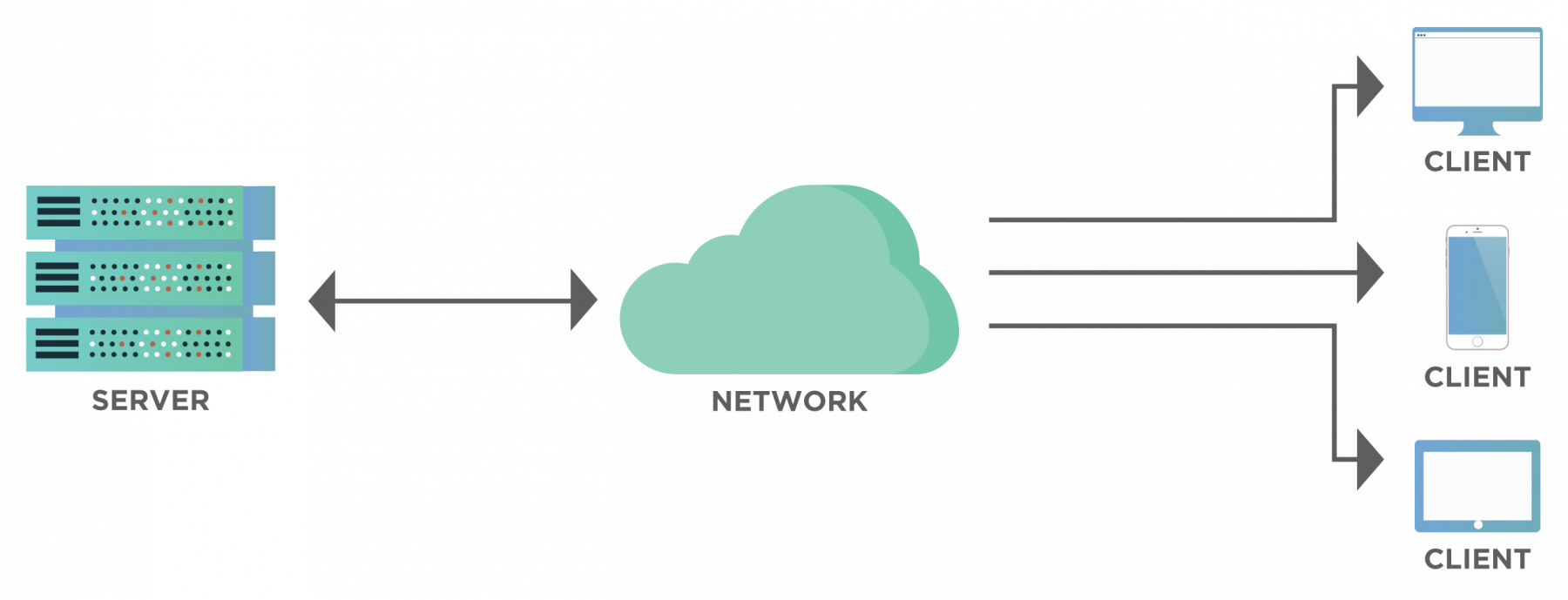
A server is a system or a computer program **that acts as the data provider**. It may provide data through LAN (*Local Area Network*) or WAN (*Wide Area Network*) using the internet. **The functionalities provided by the server are called services**. These services are provided as a response to the request made by the clients. Some of the common servers are-

* ***Database Server****- used to maintain and share the database over a network.*
* ***Application Server****- used to host applications inside a web browser allowing to use them without installation locally.*
* ***Mail Server****- used for email communication.*
* ***Web Server****- used to host web pages because of which worldwide web is possible.*
* ***Gaming Server****- used for playing multiplayer games.*
* ***File Server****- used for sharing files and folders over a network.*

These clients and servers do not necessarily be at the same location. They could either be located in different locations or may reside as different processes on the same computer. They are connected via the Web and interact via the ***HTTP protocol*** which will be discussed later in this article. There may be multiple clients requesting a server and alternatively, a client can request from multiple servers.

***Visualising the client server architecture***

The below diagram visualizes a typical client-server model-



As shown in the diagram, **a single server may serve the requests of multiple clients.** Similarly, **a client may be requesting data from different servers**. For example, consider an example of *Google*. Here, Google acts as the server and the users sitting at different places act as the clients.

Another example that we come across in our everyday life is the Online Banking Portal in which the browser that we use to open a portal acts as the client while the database and the software for banking act as the server. This propagates resource sharing across multiple users and thereby results in cost-efficiency along with time saving.

**How does the Client Server Architecture work?**

The client server model acts like a consumer-server relationship. But how does it work? Let us see some of the flow of information in a client-server architecture through a series of steps as discussed below-

1. ***The HTTP communication protocol helps establish the connection between the client and the server.***
2. ***The client sends a request in the form of an XML or a JSON over the active connection. The client and server both understand the message format.***
3. ***Upon receiving the client request, the server searches the requested data and sends back the relevant details as a response in the same format in which the request was received.***



The above diagram shows the communication process between the client and the server. The client sends an ***HTTP request***, for which the server sends an ***HTTP response***.

***Visualising the flow of client server architecture***

Assume that you need to go shopping from your home to a shop across the road. You may travel on a bicycle and on reaching can order the salesman the goods you need. Now if the salesman finds the goods, he will bring them to you else he would let you know about their unavailability.

Now, you may link the above example with the navigation to a website, the street between your home and the shop is the internet connection. The mode of transport that you chose to travel is the [***TCP/IP***](http://www.tcpipguide.com/), defining the communication protocol for the data to travel through the internet. The address of the shop is the DNS (*Domain Name Server*) of the website. Your communication language with the salesman is the HTTP (*Hypertext Transfer Protocol*) which defines the language for interaction between the client and the server. Your order request is the [***HTTP request***](https://www.toolsqa.com/client-server/http-request/) and the update on item availability is the [***HTTP response***](https://www.toolsqa.com/client-server/http-response/).

When the web browser sends a request to the server with the DNS of the website, and the server approves the client request, it sends a 200 OK success message. This message means that the server has located the website and it sends back the website files in small chunks of data to the browser. The browser then collects and assembles these small chunks to form the complete website and displays it to us.

**What are the different types of Client-Server architectures?**

Client-server architecture has the following four types -

1. *1-Tier Architecture*
2. *2-Tier Architecture*
3. *3-Tier Architecture*
4. *N-Tier Architecture*

***1-Tier Architecture***

In 1-tier architecture, **the business logic, data logic, and the user interface all reside on the same machine**.

The environment is simple and cheap because **the client and the server lie on the same system**, but the data variance leads to the repetition of work. Such systems store data in a local file or a shared driver. Examples of 1-tier applications are the **MP3 player** or the MS Office files.

***2-Tier Architecture***

The 2-tier architecture provides the best environment in terms of performance due to the absence of any intervening server.

**The user interface resides on the client side while the database on the server side. The database and business logic can be stored either at the client or the server end, but they must remain unchanged.**

If both reside at the client end then the architecture is called ***fat client thin server architecture***. On the contrary, if both reside at the server end, the architecture is called ***thin client fat server architecture***. An online ticket reservation system generally uses a 2-tier architecture.

***3-Tier Architecture***

The 3-tier architecture involves a middleware used for interaction between the client and the server. Though it is expensive but is very easy to use. The middleware improves performance and flexibility. It stores the business and the data logic. The three layers in the 3-tier architecture are-

* *Presentation Layer(Client tier)*
* *Application Layer(Business tier)*
* *Database Layer(Data tier)*

**Almost all web applications are examples of 3-tier architecture**.

***N-Tier Architecture***

The n-tier architecture is the scaled form of 3-tier architecture. In such an environment, the processing, data management, and presentation function are isolated in different layers. The isolation makes the system easy to manage and maintain. This is also referred to as multi-tier architecture.

**What are the advantages of using Client-Server Architecture?**

Client-server architecture has the following advantages -

* *It maintains data at one central location.*
* *It provides backup and data recovery options.*
* *Accessing the data from a single server help in cost-efficiency with less maintenance.*
* *The capacity of the client and the server can be individually modified.*

**Issues of Client-Server Architecture?**

Client-server architecture has the following disadvantages -

* *A virus can easily attack a client present on the server.*
* *In case of network failure, the entire architecture of an application can suffer.*
* *Man-in-the-middle attack or data spoofing is possible during data transmission.*
* *Due to vulnerability to different kinds of attacks, it requires a special and secure network operating system.*
* *Possibility of loss of data packets either completely or modification because of some intrusion during the transmission.*

**Key Takeaways**

So, what are the takeaways from this article?

* *The client server model is similar to a salesman-customer relationship.*
* *The communication between the client and the server facilitates through the HTTP protocol.*
* *The client-server architecture is categorized into 1-tier, 2-tier, 3-tier and n-tier.*
* *We discussed the various advantages of the client-server model including its centralized framework.*
* *We then saw the issues with the architecture as to how the client-server model is vulnerable to different attacks.*

The world wide web has become too complex today. With hundreds and thousands of electronic media talking to each other, the process of communication has become too complex and heavy on computation for sure. Such scenarios call for something that can ease out the communication and keep the coding complexity to a minimum as well.  That something is the focus of this post and is termed as REST.

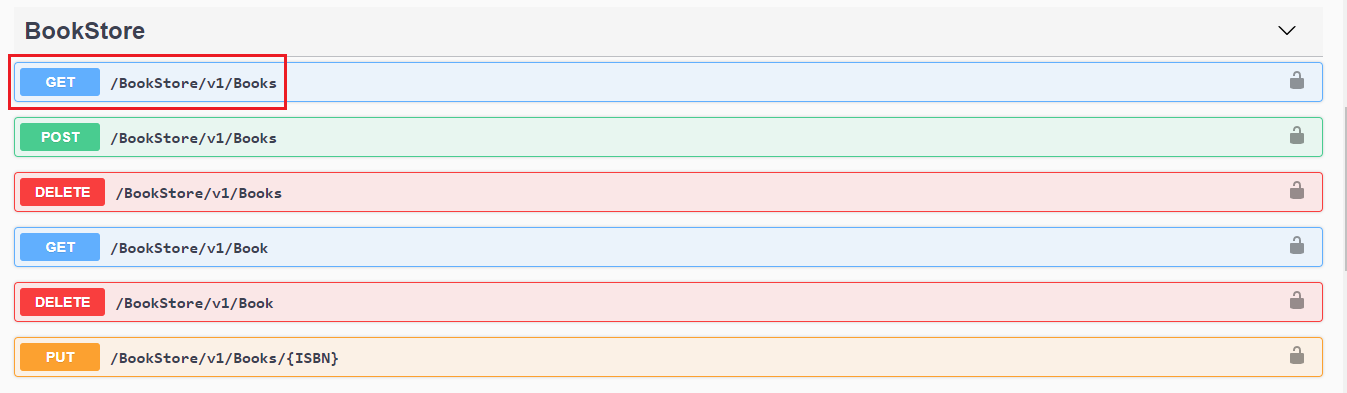
REST, or **RE**presentational **S**tate **T**ransfer, is an architectural style and provides standards between the systems on the web. Popularly, RESTful systems use the REST style to communicate with each other. So in many places, you might see ***"RESTful"*** over the internet and it means the same thing. In this article, we dwell in more details about REST basics and will cover the following topics in this article.

* *What is REST - Representational State Transfer?*
* *What are Clients and Resources?*
* *Guiding principles or constraints of REST*
  + *Uniform interface*
  + *Client-server separation*
  + *Stateless*
  + *Layered system*
  + *Cacheable*
  + *Code-on-demand*

**What is REST - Representational State Transfer?**

A web application developed using REST *( RESTful web application)* exposes the data or information about its resources which can be anything that the developer wants. This allows the client using this application to take action on the resources. For example, using information exposed to users, clients can create a new user.

As another example, suppose we have a pet store and the information related to all pets in the store is stored on the server



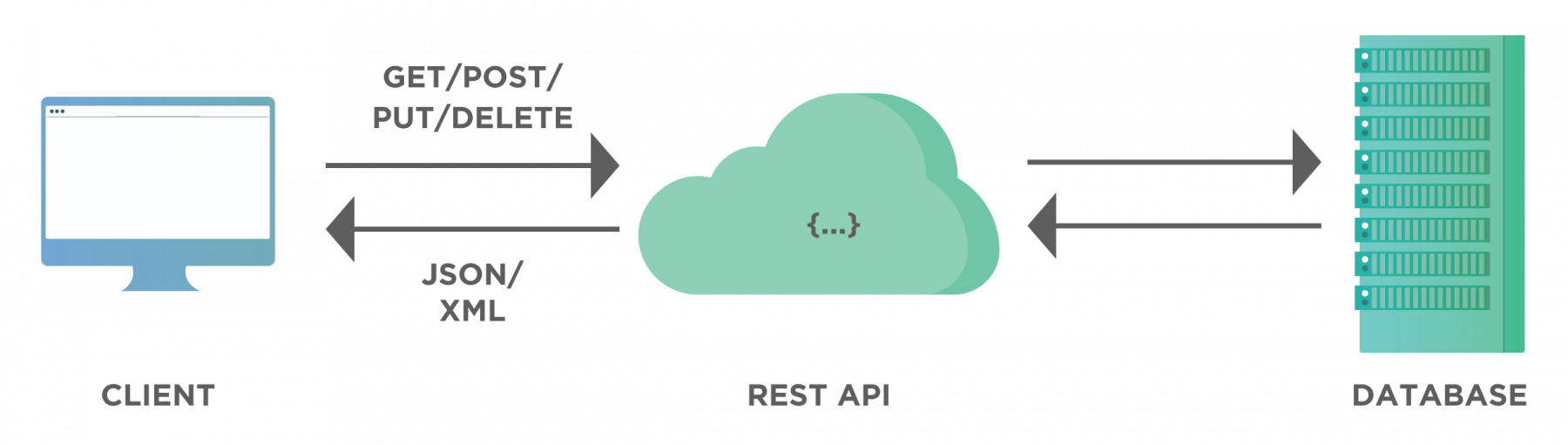
All the methods *(coloured buttons)* shown in the above screenshot correspond to REST APIs that are executed with the click of a button. In general, all this is done in the back-end and the client is not shown any of this information. Coming back, when we click the ***"GET"*** button *(/Books),* we will get the response in JSON format that will show us the details of the particular book. This response is shown below:



*Note: When we click on the****"GET"****button, in the browser we see the link changes to*[***"https://demoqa.com/BookStore/v1/Books".***](https://demoqa.com/swagger/#/BookStore/BookStoreV1BooksGet)

***Representation of REST flow***

Now let us depict the actual REST data transfer in the above example in the diagram below.



As the above diagram shows, the API works as a medium of communication between a database *(that is a part of a server)* and a client. When a client sends data through APIs, it goes to the database, do the appropriate operation *(such as add, delete, modify etc.)* and return the response data that contains response code, header files, cookie info etc.

We can summarise the REST characteristics and the working in the above diagram as follows:

* *A client access data from the server passed by REST. This can either be through authorized access or without following any strict guideline.*
* *As we can see, the application developed using REST is an interactive application and mostly it uses web services. In this case, the web service follows RESTful guidelines and fulfils the constraints of REST that are discussed later in this article.*
* *A web service using REST provides web resources in textual format and allows them to be read and edited using a predefined set of operations.*

REST is a way to access a web service and is often viewed as an alternative to [***SOAP***](https://www.toolsqa.com/soapui/what-is-soapui/) *(Simple Object Access Protocol).*

In a RESTFul application, we have entities namely client and resource which are used commonly. Let us discuss them briefly next.

**What are Clients and Resources?**

**A client can be a software or a person or a system using the APIs to access data from another application server**.

For example, a developer might access Facebook APIs to embed a live post in their own website. The developer program will call the Facebook API through a browser. So in this case, the browser acts as a client that is calling the Facebook APIs.

If we visualize this system using the REST diagram above, the client or browser will connect to Facebook-Server over REST API and then get the information required to render it on the screen.

***What Is A Resource in REST?***

**Any smaller unit that can be transformed and addressed through a URL and HTTP method is considered a resource**.

This resource makes changes to the database.

For one application, you may have a lot of resources with all of them assigned a particular task. For example, an online book store may have a resource as a table of the database. A resource in a REST architecture is anything that a client has access to and can modify or update.

**Guiding principles or constraints of REST**

For an API to be RESTful, it has to fulfil or adhere to the following guiding principles or constraints defined by REST.

***Uniform interface***

The uniform interface principle has the following parts that an API has to follow:

1. *The request to the server needs to have a resource identifier.*
2. *The server returns the response and includes information such that the client can modify or edit the resource.*
3. *The request sent through the API contains the information that the server needs to execute the request. Each response returned by the server also contains all the information so that client understands the response.*
4. *Hypermedia as the engine of application state. The application means the web application that is running on the server. Hypermedia is the hyperlinks or simple links included in the response. So the server basically informs the client about the ways to change the application web.*

***Client-server separation***

The interaction between the client and the server is independent and is only in the form of requests. The client initiates a request and the server sends the responses. The response is a reaction to the request. So all server does is wait to receive requests from the client. It never sends out information about the resources to the client on its own. For more details on client-server refer to our tutorial, [***Client-server Architecture.***](https://toolsqa.com/client-server/client-server-architecture-and-model)

***Stateless***

The word ***"stateless"*** means the server does not keep track of the user who uses the API. So when a new request comes in, the server will not know if the same user has sent a GET request in the past for the same resource. It doesn't remember the resources requested by the user earlier. For example, HTTP is a stateless protocol. HTTP server does not keep track of any state of information passed to and fro. Hence at any given time, a client can send a valid command, and the server will not relate or keep track of any previous similar commands.

Hence each request regardless of the other requests made by the user will contain all the data needed by the server to execute the request and send a response.

***Layered system***

A layered system provides a hierarchical structure between a client and a server. There can be a lot of intermediaries between the client and the server working along with REST API without the client's notice. Our clients think there is a direct connection to the server. We then take advantage of it to improve our architecture and bring down our distributed system complexity. These intermediate elements provide a security layer, load-balancing layer, and other functionality to the system. The only guideline is that the presence of these intermediate layers should not affect the request or response.

*Note: The abstraction of layers does not let one layer be aware of the presence of another layer.*

***Cacheable***

The server data received in the response contains information regarding whether the data is cacheable or not. If the data is cacheable, it will contain some kind of version number that makes caching possible. The client will know which version of the data it has got from the previous response. This way client can avoid requesting repeated data. Cacheable data *(and therefore version number)* also helps the client to know the expiration of version data and the requirement of a new request to fetch the latest data.

***Code-on-demand***

This particular constraint ***"Code on demand"*** is optional and without fulfilling it we can have a RESTful API.

The client can send a request to the server asking for the code and then the server will respond with some code in the form of a script or some other entity. For example, servers can extend the client functionality by downloading and executing pre-compiled code like an applet or a client-side script like JavaScript. So when we click on any video on Facebook, Facebook will run a precompiled or any third-party software to run that video.

Once an API fulfils the above constraints we discussed, we can say it is a RESTful API.

**Key Takeaways**

In this article, we discussed the introduction of the REST protocol for web services.

1. *The REST (Representational State Transfer) is a standard that operates between machines mostly executing web services.*
2. *The server receives a request in form of REST API. The server then runs the web application and sends a response to the client.*
3. *For an API to be a valid REST API (RESTful API) it needs to fulfil some constraints that we discussed above.*
4. *A RESTful API adheres to these constraints (also called guiding principles).*

*.*

# What is REST Assured?

Rest-Assured is an open-source Java-based library useful for testing and validating [***REST APIs***](https://www.toolsqa.com/rest-assured/rest-api-end-to-end-test/) or RESTFul Web Services. It is like a headless (no GUI) client that accesses and tests the REST Web Services.

Rest assured simplifies the testing of REST-based services. It brings out the simplicity of dynamic languages like Groovy and Ruby that perform API testing in Java. The library supports any HTTP method and also has explicit support for GET, POST, PUT, DELETE, OPTIONS and HEAD. It also includes specifications and validation like parameters, headers, cookies, etc. We can also use it to verify and validate the response of the HTTP requests.



[***Source***](https://devqa.io/)

Apart from testing JSON-based web services, Rest Assured can also be used to test XML-based web services.

We can also integrate this library with [***JUnit***](https://www.toolsqa.com/java/junit-framework/junit-introduction/) and [***TestNG***](https://www.toolsqa.com/testng/testng-tutorial/) frameworks and write test cases for applications.

Further, it can be integrated well with [***Maven***](https://toolsqa.com/maven/maven-introduction/) and its efficient matching techniques produce straightforward results.

Yet another powerful feature of REST assured is its support for XML Path and JSON Path syntax to check specific elements of the response data similar to using XPath API. For people new to such concepts, please refer to the following syntax examples.

XPath syntax:

<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book>

<title>Freedom In Exile</title>

<price>14.29</price>

</book>

</bookstore>

JSON path syntax:

$['store']['book'][3]['title']

Apart from all the above features, this fantastic library also provides various other features like ***DSL-like syntax, Specification Reuse, XPath-Validation, easy file uploads***, and all the features that are conducive to automated testing.

**Why do we need Rest Assured?**

The above discussion about this java library convinces us that it is a reliable library. But why do we need it or what are the reasons to use it in our application testing?

Following are the main reasons we need Rest Assured:

* *It is an* ***open-source library*** *and has an active development community making it a great choice for API automation.*
* *Earlier, we had to use dynamic languages like Ruby, Groovy for API testing and it was quite challenging.*
* ***Validation and testing of REST services are harder in Java. Using REST Assured, it becomes simpler and easier.***
* ***This library uses Java and therefore it becomes simple to send HTTPS requests with customizations using basic Java code****. Once we know the basics of API and*[***integration testing***](https://www.toolsqa.com/software-testing/integration-testing)*, automation using Rest Assured gives good confidence on the backend. Thus we can focus more on front-end testing.*

**Advantages of Rest Assured**

The following table lists some of the advantages of the library.

| **SNo** | **Advantages** |
| --- | --- |
| 1 | It is open-source and hence free to use. |
| 2 | It is very rich in syntax and ready-made assertions. Rest Assured requires less coding as compared to Apache HTTP Client. |
| 3 | The setup of Rest Assured is easy and straightforward. |
| 4 | The response is given in JSON or XML format and is easy to parse and validate. |
| 5 | It uses inbuilt ***[Hemcrest Matchers](https://en.wikipedia.org/wiki/Hamcrest)*** for easy extraction of values. |
| 6 | Response time is quick as also an assertion of status code. |
| 7 | The library has a powerful logging mechanism. Also, we can verify headers, cookies, content type, etc on the fly. |
| 8 | It can easily be integrated with other Java libraries like TestNG, JUnit, etc. We can also integrate it with Selenium-Java and achieve end-to-end automation. |
| 9 | It has very good support for various API authentication mechanisms. |
| 10 | It supports JsonPath and XmlPath that helps in parsing JSON and XML response. It also has support for the JSON Schema Validation library to verify JSON Schema. |
| 11 | Rest Assured can also be integrated with Maven and CICD. |
| 12 | It supports multi-part form data |
| 13 | Supports Spring Mock MVC, Spring Web Test Client, Scala, and Kotlin. |
| 14 | It follows the [***BDD (Behavioural Data-Driven)***](https://www.toolsqa.com/cucumber/behavior-driven-development/) approach and keywords like given() when(), then() which makes code readable and supports clean coding. This feature is available from version 2.0. |
| 15 | REST Assured 4.1.2 adds support for Java 13. |

**Disadvantages of Rest Assured**

The library has the following disadvantages.

1. *It does not support the testing of*[***SOAP(Simple Object Access Protocol)***](https://www.toolsqa.com/soapui/soapui-tutorial/)*APIs explicitly.*
2. *Using this library requires that the user has good Java programming knowledge*
3. *There is no inbuilt reporting in Rest Assured.*

**Key Takeaways**

We have the following takeaways from this post:

1. *It is an open-source Java library and is free to use.*
2. *It is used for validating and testing Java APIs.*
3. *Apart from other features like Java DSL syntax, also supports all HTTP methods and REST methods like GET, POST, PUT, DELETE etc.*
4. *We can integrate Rest Assured with all major automation frameworks like TestNG, JUnit as well as integrate it with maven and CI/CD.*
5. *The library has a rich syntax and since it is open-source more and more functionality keeps on adding which makes it a very efficient and simple library for API automation.*

# What are the prerequisites to configure Eclipse with Rest Assured?

The first step towards setting up the development environment so that we can test some RESTful APIs is to evaluate the prerequisites. The following list highlights the required software for setting up Eclipse with rest-assured. Before proceeding, please make sure they are installed on your system.

* *Latest Java version.*
* *Working Eclipse IDE.*
* *TestNG setup.*

If any of the above is not present on the system, then we have to follow the links given below :

1. [***Set up Java***](https://toolsqa.com/selenium-webdriver/install-java/)
2. [***Setting up Eclipse***](https://toolsqa.com/selenium-webdriver/download-and-start-eclipse/)
3. [***Set up TestNg***](https://toolsqa.com/testng/install-testng/)

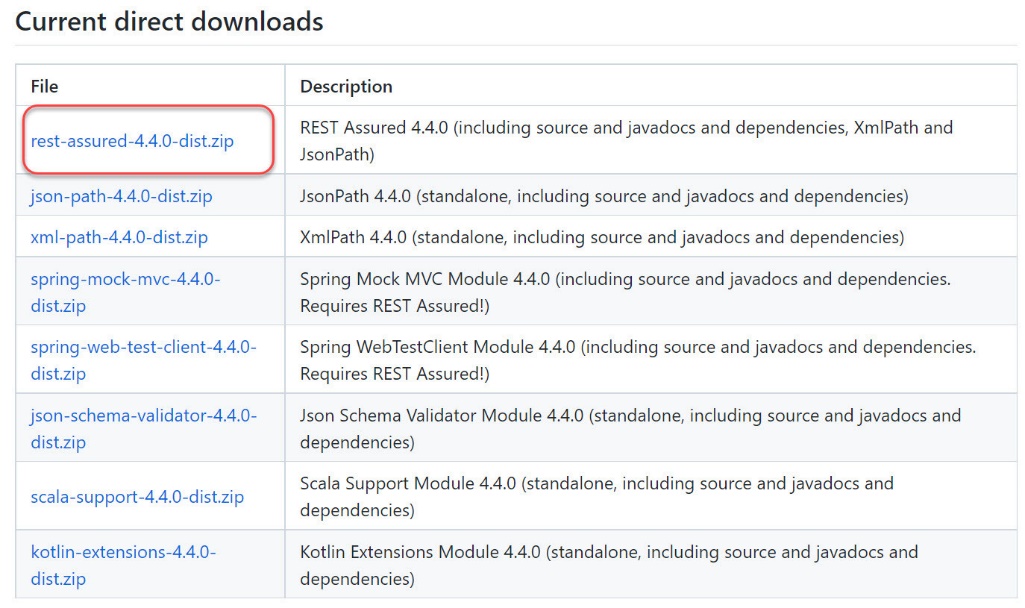
Once the prerequisites are up and running we can proceed with the rest assured set up on Eclipse.

**How to download rest-assured jars in Eclipse?**

The rest assured jar files can be downloaded from the following link.

[***https://github.com/rest-assured/rest-assured/wiki/Downloads.***](https://github.com/rest-assured/rest-assured/wiki/Downloads)

When we navigate to this page, we can see the section “***Current direct downloads***”. Go to this section and click on the link to the ***dist package***.



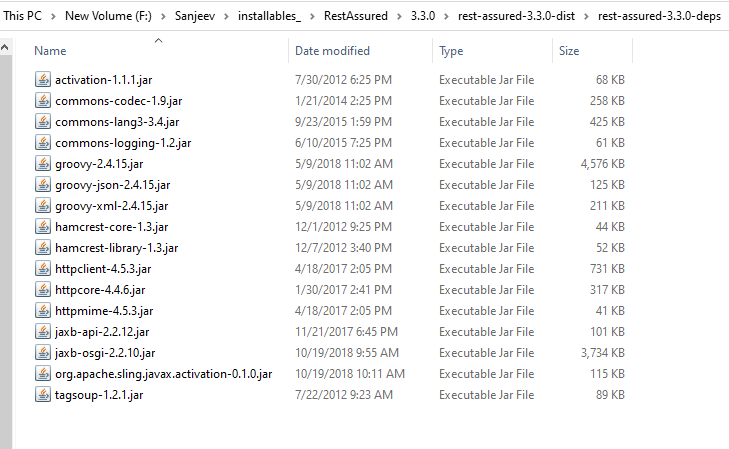
***Note****: At the time of this writing Version 4.4.0 is available for download.*

Once the link is clicked, the browser will download the ***rest-assured-4.4.0-dist.zip*** file on the system (*mostly in the "****downloads****" folder for Windows OS*). Navigate to the folder where the zip file was downloaded and simply unzip the files.

Now go to the folder where the files were just unzipped and open it. The contents should be the following in this folder.

* ***docs****: A folder containing javadocs.*
* ***rest-assured-3.0.3.jar****: jar file that contains rest-sssured classes.*
* ***rest-assured-3.0.3-deps.zip****: Another zip file containing all the dependency jars.*

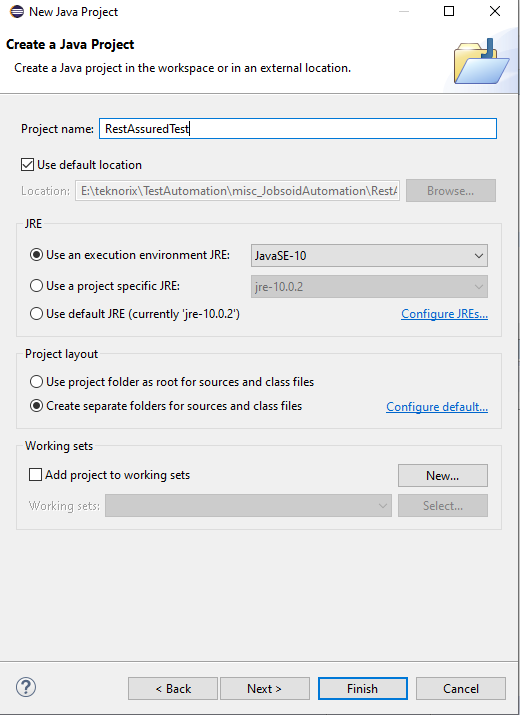
*Note: We have to also unzip the rest-assured-4.4.0-deps.zip dependency jars. In the end, your folder would look like this.*



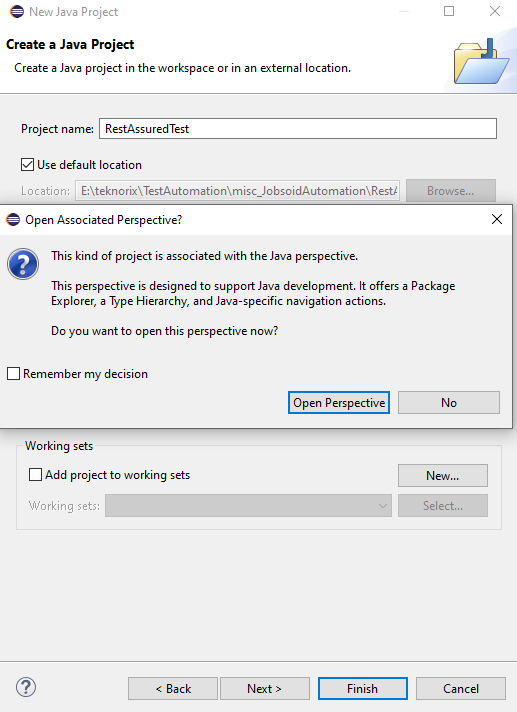
Now that we have rest assured jars package ready,  we can go ahead and create a project in Eclipse that will use rest assured.

**Creating a project in Eclipse for rest assured**

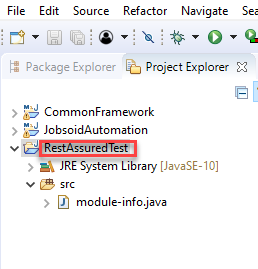
To add and use unzipped rest assured jars in an actual project, we have to first create a new project in Eclipse. So we create a project, let's say, RestAssuredTest as shown below.



Once clicked on the ***Java Project*** link, a new project window opens up. Enter the name of the project as ***RestAussuredTest*** and click on the ***Finish*** button. Once ***Finish*** is clicked, we get the following screen. Click Open Perspective. (*Alternatively, if we have already checked the option "****remember my decision****", then this step will not be seen*)



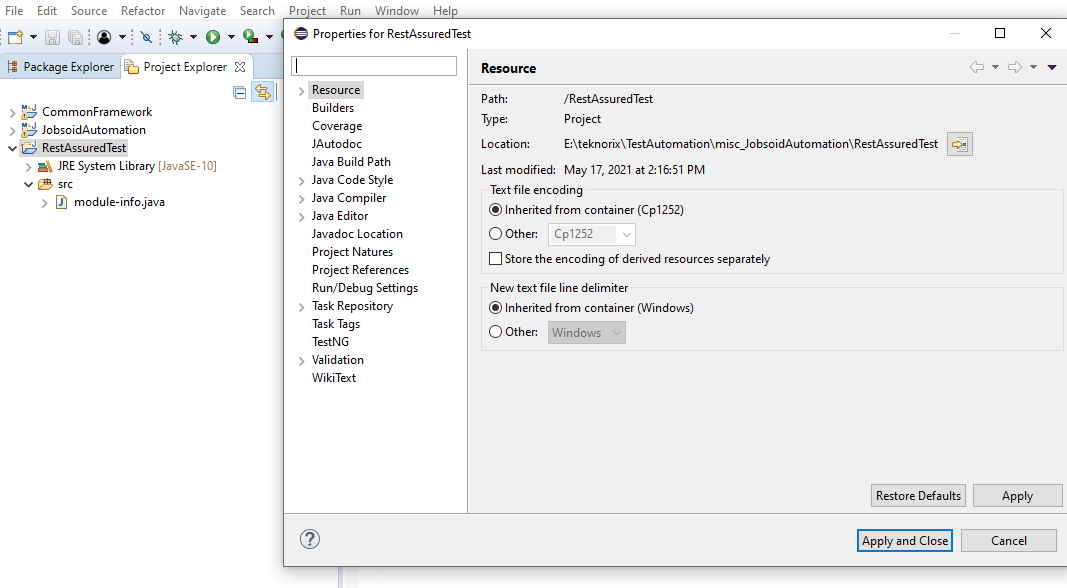
Now the project with the name RestAssuredTest should display in the package explorer.



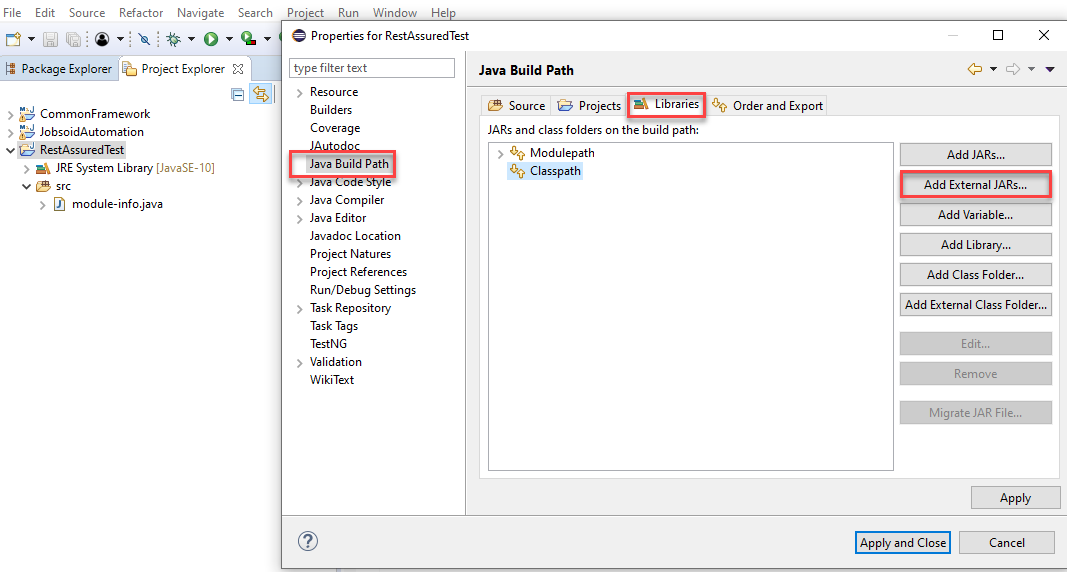
Once the project is created in Eclipse, now it is required to add the unzipped jars in the classpath of the Eclipse project.

**How to setup rest-assured jars in the class path?**

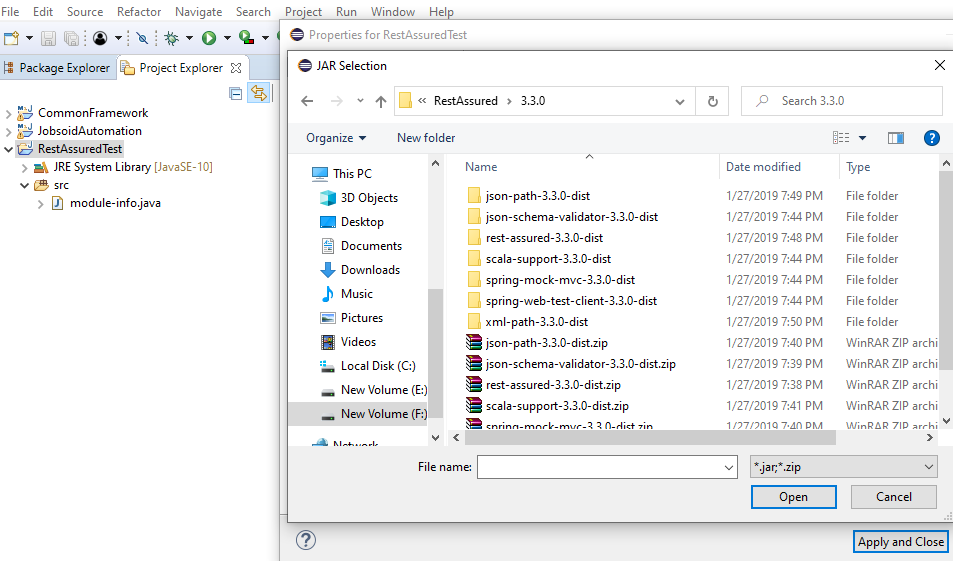
Select the Java project folder we just created and right-click on the project folder in the package explorer pane and choose ***Properties***. This will open up the project properties pop-up window as shown below.



Select  ***Java build path*** option in the left pane of the properties window. In the Java build path pane, on the left-hand side, you will see the ***Libraries*** pane.



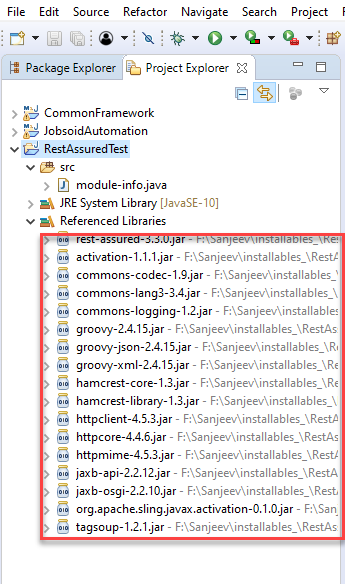
Here we are going to reference the rest assured jar files that we downloaded earlier. To reference the jars,  click on the "***Add external jars***" button and navigate to the folder where we unzipped all our rest-assured jars. Refer to the image shown below.



Now include the following Jars

* *rest-assured-3.3.0.jar*
* *All the jars in the folder rest-assured-3.3.0-deps*

With this, we have successfully set up a Java project in Eclipse with a rest-assured library. If we now check the project explorer, we will get the following view.



Now we are ready to use Rest Assured features in Eclipse.

**Key TakeAways**

In this article, we configured the Eclipse project with Rest Assured Library.

* *Rest Assured library comes in the form of jars which we can download freely from GitHub and unzip on our local machine.*
* *The prerequisites for using Rest Assured jars are we should have Java, Eclipse, and TestNG properly configured and running on the machine.*
* *We can then create a Java project in Eclipse and associate the Rest Assured jars with it by specifying the jar path in the Build path.*

With these steps, we can configure Rest Assured with the Eclipse project. In the next article, we will create a simple REST API with the Rest Assured library.