Table of Contents

[Chapter 1 | Installation, ENV Configuration, JMeter Test Recording Mode 4](#_Toc5880043)

[Download JMeter 4](#_Toc5880044)

[Installing JDK 4](#_Toc5880045)

[Setting Java environment 4](#_Toc5880046)

[Running JMeter 4](#_Toc5880047)

[JMeter Classpath 4](#_Toc5880048)

[Configuring the proxy server 4](#_Toc5880049)

[Running in Non-GUI Mode 4](#_Toc5880050)

[Running in Server mode 4](#_Toc5880051)

[Recording Script using HTTP Test Recorder 4](#_Toc5880052)

[Recording script using Chrome Brower Extension 4](#_Toc5880053)

[Converting HTTP Web Archives (HAR) to JMeter test plan 4](#_Toc5880054)

[Executing JMeter test 4](#_Toc5880055)

[Analyzing Test Result 4](#_Toc5880056)

[Chapter 2 | Anatomy of a JMeter Test 4](#_Toc5880057)

[Test Plan 4](#_Toc5880058)

[Test Fragments 4](#_Toc5880059)

[Configuration Elements 4](#_Toc5880060)

[Chapter 3 | Working on Forms 5](#_Toc5880061)

[Chapter 4 | Managing Sessions 5](#_Toc5880062)

[Managing session with cookies 5](#_Toc5880063)

[Managing session with URL Rewriting 5](#_Toc5880064)

[Chapter 5 | Handling Responses 5](#_Toc5880065)

[Regular Expression Extractor 5](#_Toc5880066)

[Regular Expression Tester 5](#_Toc5880067)

[Chapter 7 9](#_Toc5880068)

[Pre-Processor 9](#_Toc5880069)

[HTML Link Parser 9](#_Toc5880070)

[HTML URL rewriting Modifier 9](#_Toc5880071)

[User Parameters 9](#_Toc5880072)

[Post-Processors 9](#_Toc5880073)

[Regular Expression Extractor 9](#_Toc5880074)

[XPath Extractor 9](#_Toc5880075)

[Result Status Handler 9](#_Toc5880076)

[Debug Post Procesor 9](#_Toc5880077)

[Chapter 8 – Timers 9](#_Toc5880078)

[Beanshell Timer 9](#_Toc5880079)

[BSF Timer 9](#_Toc5880080)

[Constant Timer 9](#_Toc5880081)

[Gaussian Random Timer 10](#_Toc5880082)

[Uniform Random Timer 10](#_Toc5880083)

[Poisson Random Timer 10](#_Toc5880084)

[Synchronizing Timer 10](#_Toc5880085)

[Constant Throughput Timer 10](#_Toc5880086)

[Custom Timer – gets added from google plugin 10](#_Toc5880087)

[Chapter 9 – Assertions 10](#_Toc5880088)

[Duration Assertion 10](#_Toc5880089)

[Size Assertion 10](#_Toc5880090)

[XML Assertion 10](#_Toc5880091)

[XML Schema Assertion 10](#_Toc5880092)

[HTML Assertion 10](#_Toc5880093)

[Response Assertion 11](#_Toc5880094)

[XPath Assertion 11](#_Toc5880095)

[Compare Assertion 11](#_Toc5880096)

[Chapter 10 – Config Elements 11](#_Toc5880097)

[Counter 11](#_Toc5880098)

[Random Variable 11](#_Toc5880099)

[FTP Request Default 11](#_Toc5880100)

[Login Config Element 11](#_Toc5880101)

[HTTP Request Default 11](#_Toc5880102)

[Java Request Default 11](#_Toc5880103)

[User Defined Variables 11](#_Toc5880104)

[CSV Data Set Config 11](#_Toc5880105)

[HTTP Authorization Manager 11](#_Toc5880106)

[HTTP Cookie manager 11](#_Toc5880107)

[Chapter 20 – JMeter Inbuilt Functions 11](#_Toc5880108)

[Pointers 12](#_Toc5880109)

[Performance Testing Strategy 14](#_Toc5880110)

## Chapter 1 | Installation, ENV Configuration, JMeter Test Recording Mode

## Why JMeter?

It was chosen for several reasons:

* open source, free tool, easy for installation and updating
* supports multi types protocols
* ability to record tests scenarios for different testing targets: Load Testing, Distributed Testing, Functional Testing, Stress Testing and Scalability Testing.
* intuitive GUI for the first acquaintance
* ability to make manual and automatic scripts recording
* it has a lot of tutorials and constant support
* many visualization reports (charts, tables, tree, for overall view of tests), understandable for development team and customer
* highly extensible
* JMeter is an independent java desktop application that’s why it can be run on multiple platforms
* can be integrated with Selenium, SOAP, Blazemeter, Jenkins, BadBoy, etc.

As you can see, Apache JMeter has many advantages over similar tools, but in our daily work we apply to it because it enables to execute load and performance tests for application at the different server types that include Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3.

## How to use Apache JMeter for https performance tests?

At first you should install Apache JMeter (version 2.10 or later, in this case we have 2.13), recording will be more effective with Oracle Java 7 or later and browser Mozilla Firefox. In our daily work we use HTTP(S) Test Script Recorder for recording HTTP and HTTP(S) requests in Tests plans which could be ran all the time during web application is under testing. In this article you can find steps for Test plan creation for web application with https protocols and it includes csrf tokens.

## How to create Test plan?

Start Apache JMeter, at first you should add Threats Group to save recording Test plan in.

Steps: Right mouse click on Test Plan/Add/Threads (Users)/Thread Group (1 element pic.1)

For displaying all results after performance tests running you should add reports view. Choose any Report in the list of Listeners.

Steps: Test Plan/Add/Listener/View Result Tree (2 element pic.1)

Add HTTP Cookie Manager to Thread Group to define additional cookie.

Steps: Right mouse click on Thread Group/Add/Config Element/HTTP Cookie Manager (3 element pic.1). Pic.1. Thread Group creation

## How circumvent https and record Test Plan automatically?

Add HTTP(S) Test Script Recorder to WorkBench

Steps: Right mouse click on WorkBench/Add/Non-Test Element/HTTP(S) Test Script Recorder (1 element pic.2)

Pic.2. Recording Test Plan

For bypassing bmp, js, gif etc. elements that are not relevant to test execution press 'Add suggested Excludes' button (2 element pic.2). Check that URL Patterns to Exclude were added (3 element pic.2). And remember that before start recording you should choose Target Controller. From drop-down menu you should choose Test Plan > Thread Group for saving recording plan to Threat Group (4 element pic.2).

## How to change Proxy configuration for recording tests?

We create and run tests using proxy server. Change Proxy configuration in browser. For tests recording use exclusively Mozilla FireFox browser because some other browsers (like Chrome, etc.) do not allow you to override system-wide configuration for their proxy settings. Open FireFox browser. Change proxy configuration

Steps: find tab 'Edit'/Preferences/Advanced/ tab Network /Settings.

At Connection settings window:

* choose ‘Manual proxy configuration’
* add ‘localhost’ to input ‘ HTTP proxy’
* add ‘8080’ to input ‘Port’.
* choose ‘Use this proxy server for all protocols’ and press ‘Ok’ button.

Check port at Test script Recorder, it should be 8080 (5 element pic.2) the same as in your FireFox browser.

Choose HTTP(S) Test Script Recorder, click 'Start' button (6 element pic.2). On the bottom of the Script Recorder window, JMeter proxy server will start and be used to intercept and record browser requests. If you are trying to record test at first time, it will display an error message that it cannot start because a certificate does not exist. Click ‘OK’ button, then click ‘Start’ one more time. You should find a message that says that a temporary certificate named Root CA certificate: ApacheJMeterTemporaryRootCA.crt created in JMeter bin directory (pic.3). Root Certificate CA was created and saved at Apache JMeter's 'bin' folder. The root certificates are used whenever you connect via an https connection to make sure that you are connecting to who you think you are. In our case we should use our own created certificate authority to issue certificates for internal web servers. Just click OK and try to continue recording actions in FireFox browser.

Pic.3. Created Root CA certificate

Open application that will be tested at FireFox browser and make test actions step by step. Use valid credentials for authorization. All steps made by user will be recorded to Test Plan set in JMeter. After making all tests actions stop recording.

Steps: open Apache JMeter/HTTP(S) Test Script Recorder/press 'Stop' button (7 element pic.2).

As a result all actions that user makes in application were recorded to Thread Group (1 element pic.4) .

Pic.4. Test scenario in JMeter. Tests Properties

If some actions were needless you can delete requests from created Test plan clicking right mouse button on excess request and finding Remove item.

Choose Thread Group and change test name (2 element pic.4). For loading web application properties at the Thread Group can be changed number of users, ramp-up, loop count can be chosen) (3 element pic.4).

If the goal is reached and result satisfies you, do not forget to save Test plan via File/Save Test plan as… after tests were saved, you may add them to any test plan that you have open by using the "Merge" menu item, and selecting your saved WorkBench. You can work with it via other testing tools like Soap, Blazemeter , etc. just download saved .jmx file.

For running tests push 'Start' button at the top menu (4 element pic.4) and find result at View Result Tree (5 element pic.4)

## What should you do with requests that do not run due to csrf tokens?

In this application there are some csrf tokens present and due to them we have some errors with code 403 (1 and 2 elements pic. 5). At test result view you can find Request and Response Data.

Pic.5. Tests result in JMeter

For successful testing of this web application we need to intercept csrf tokens value and put it to our request as variable values Open login request (1 elements pic. 6) and find post data (login, password and csrf token) (2 elements pic. 6).

Pic.6. Put csrf token to HTTP request as variable value

For bypassing csrf tokens value should be captured by Regular Expression Extractor and input to POST requests. Duplicate first test page at the Test Plan (using right mouse click). In our case it would be request 160/ (3 elements pic. 6). Rename it (in our case we rename it as ‘160/ token’ ), we have two similar HTTP(S) requests (4 elements pic. 6).

First HTTP(S) request for getting csrf token from the body of GET request via Regular Expression Extractor. Steps: find 160/ token, right mouse click Add/Post Processors/Regular Expression Extractor) (5 elements pic. 6).

Put data to Regular Expression Extractor inputs (1 element pic.7).

Pic.7. Get csrf token from the GET request

At 'View Result Tree' (1 element pic. 8) on the first test page 160/ (2 element pic. 8) find Response Data of the first request that gets csrftoken value (3 element pic.8)

Pic.8. Find tokens value in Response Data of GET request

At Regular Expression Extractor (1 element pic. 9) add csrf token line 'name='csrfmiddlewaretoken' value='xfgTXpJN5vkR4po4urpZI8tIyiuBrNvw'' to input 'Regular Expressions' Change csrf token value to variable value. Using (.+) construction (2 element pic.9).

Pic.9. Construction for changing csrf token value to variable value

Find login POST request (1 element pic.10), find ‘csrfmiddlewaretoken’ (2 element pic.10). Change csrfmiddlewaretoken to variable value using ${token} contraction (3 element pic.10).

Pic.10. Put csrf token to HTTP request as variable value for login

Run Test Plan and check result at View Results Tree (1 element pic.11). Login pass successfully (2 element pic.11).

Pic.11. Successful login result

Check Tasks marked red and find response data with error 403 (3 element pic.11). In this requests with errors (1,3,4,5,6 elements pic.12) you should put variable value to csrf tokens (2 element pic.12) at PUT requests with errors.

Pic.12. Put variable value to all HTTP request with csrf tokens error

Try to run Test plan and check that all test’s requests were passed.

If an application has high security, csrf tokens will always be refreshable in all requests. That’s why you need to put Regular Expression Extractor to all tasks that were not passed due to variable tokens (Duplicate GET requests and put variable value to csrf tokens at PUT requests). In our case we take first ‘take’ request in which we find value of scrf tokens - duplicated it and implemented Extractor

Pic.13. Add additional Regular Extractor

Try to run Test plan and check that all tests requests were passed.

Pic.14. Successful login and logout

In our case two Extractors (1, 2 elements at requests 3, 4 elements pic. 14 ) give ability to pass four requests which include csrf tokens (5-8 elements pic. 14).

For load, stress and performance testing we often used Apache Jmeter and Blazemeter together. It gives a more accurate and clear picture of user's experience and how a large number of users works with application simultaneously. Full statistics returns from server and reports through graphical diagrams, tables, etc.

### Download JMeter

### Installing JDK

### Setting Java environment

### Running JMeter

### JMeter Classpath

### Configuring the proxy server

### Running in Non-GUI Mode

### Running in Server mode

### Recording Script using HTTP Test Recorder

### Recording script using Chrome Brower Extension

### Converting HTTP Web Archives (HAR) to JMeter test plan

### Executing JMeter test

### Analyzing Test Result

## Chapter 2 | Anatomy of a JMeter Test

Anatomy of a JMeter test

**Test Plan**

This is the root element of the JMeter scripts and houses the other components, such as threads, config elements, times, Preprocessors, Postprocessors, assertions and listener. It also offer a few configuration of its own.

**Thread Groups**

- This is entry point of a test plan.

- They represent the number of threads/users JMeter will use to execute the test plan.

- All controllers and samplers for a test must reside under a thread group. Listener may remain under thread group or under test plan.

- Thread Group configuration provide options to specify the number of threads that will be used for the test plan, how long it will take for all threads to

become active (ramp up ) and the number of times test will be executed.

Controllers - Controllers drive the processing of a test and comes in two flavors -

1. Sampler controller - sampler controller sends reqest to a server. These includes HTTP, FTP, JDBC etc.

List of sampler controllers -

1.1 HTTP request

1.2 JDBC Request

1.3 LDAP Request

1.4 SOAP/XML-RPC Requst

1.5 WebService (SOAP) reqest

1.6 FTP Request

2. Logical Controller - This allows the customization of the logic used to send the requets. Like loop controller can be used to repeat an operation a certain number of times. if controller is for selectively executing a request. Helps customzie the logic used to decide how requests are sent to server. They can modify requests, repeat reqest, interleave request, control the duration of request execution, switch request, measure the overall time taken to perform request. it is like a container which holds steps.

List of logical controller -

Test Fragments - A spcial type of controller is used purely for code reuse within a test plan. They exist on the test plan tree at the same level as the thread group element and are not executed unless referenced either by any include or module controller.

Listeners

Timers

Assertions

### Test Plan

#### ThreadGroup

#### Controllers

#### Sampler

#### Logic controller

### Test Fragments

#### Listeners

#### Timers

#### Assertions

### Configuration Elements

#### Preprocessor

#### Postprocessor

## Chapter 3 | Working on Forms

## Chapter 4 | Managing Sessions

### Managing session with cookies

### Managing session with URL Rewriting

## Chapter 5 | Handling Responses

### Regular Expression Extractor

### Regular Expression Tester

### CSS/JQuery Extractor

### XPath Extractor

### XML Extractor

### JSON Extractor

### HTML Extractor

**Performance Testing -** performance testing is in general, a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

**Responsiveness** - response of application

**Throughput** - request per second, number of unit of work in a unit of time

**Reliablility** - should not throw denial of service

**Scalibility** - should have ability to handle extra work load without impacting performance

**Assertion** - are required for functional testing

**Listener** - are helpful to view result data

**Regular Expression**

. any number of characters

+ at least 1 character

[] - range of number / alphabet

[a-z] - any alphabet from a to z

[0-9]\* - any number of digit

<a class="theme-nav-link" href="/schedules" title="click to go to 'Schedules'" target="\_self" data-identifyelement="18">Schedules</a>

## Chapter 7

### Pre-Processor

### HTML Link Parser

### HTML URL rewriting Modifier

### User Parameters

### Post-Processors

### Regular Expression Extractor

### XPath Extractor

### Result Status Handler

### Debug Post Procesor

http://192.168.1.2:1080/WebTours/index.htm

## Chapter 8 – Timers

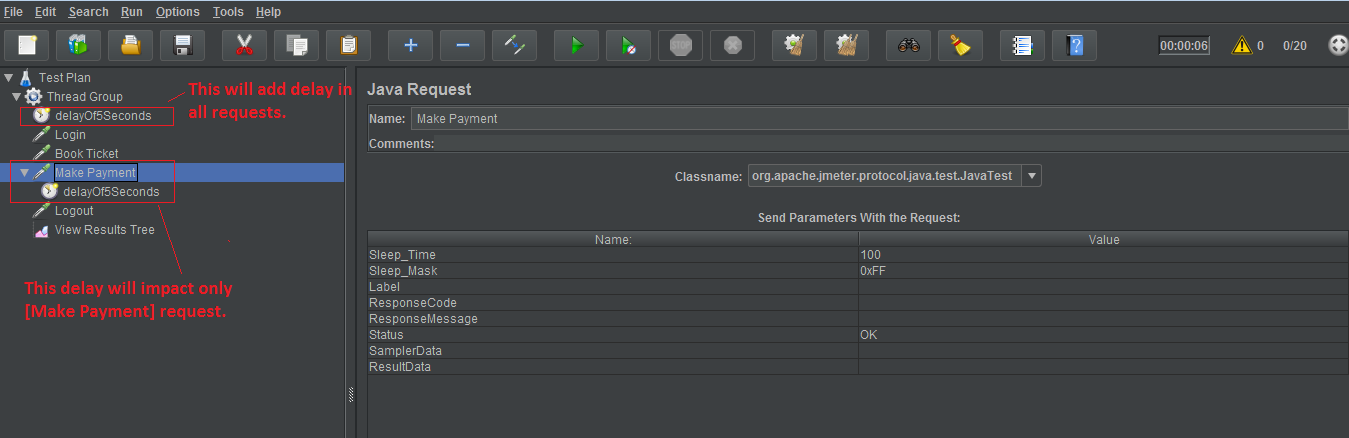
To make our script realistic we have to add some delay between two request called as think time. We can add delay in JMeter using Timer.

### Beanshell Timer

### BSF Timer

### Constant Timer

It add fix time delay between each request. Constant timer will add delay in all request in the scope. If you want to add delay in a particular request, add constant timer as a child of that request.



If you add two constant timer in same scope, sum of both timer will get added as delay.

### Gaussian Random Timer

We have 3 random timer, Gaussian, Uniform & Poisson. This timer follows distribution of time as like bell curve.

### Uniform Random Timer

### Poisson Random Timer

### Synchronizing Timer

This timer blocks the thread until a given number is reached. Then all threads are released at once. This helps in creating large number of threads on server.

In case value zero is passed into timer, by default it will take 100 threads. It is mandatory to provide value else JMeter shows error.

Helps in simulating peak load on server.

We can use random() function to create random number of thread for each request.

### Constant Throughput Timer

It adds variable pauses to calculate a possible given number.

### Custom Timer – gets added from google plugin

## Chapter 9 – Assertions

JMeter comes bundled with assertion components. Assertion are used to perform additional checks on samplers and are processed by default after every sampler in the same scope, except for in cases whee they have been added as a direct child of a sampler. You can think of these as mini unit tests within your test plans that ensures the responses from server actually work fine. It is used in JMeter for functionality testing. We can use them for performance testing also, but they are more significant for functional testing. It inserts checkpoint or verification point on response.

For example – when you make HTTP request in a web application, the server responses may respond with different HTTP header status codes, denoting the success or failure of request. A status code

* 200, for example, means the request succeeded
* 401 means access was denied to the requested resource
* 500 means an internal server error occurred

Types of Assertions are follows –

1. Response Assertion
2. JSON Assertion
3. Duration Assertion
4. Size Assertion
5. XML Assertion
6. XML Schema Assertion
7. HTML Assertion
8. XPath Assertion
9. Compare Assertion
10. Bean Shell Assertion
11. JSR223 Assertion
12. MD5Hex Assertion
13. SMIME Assertion

### How it works

### Duration Assertion

It measures time take by samplers. Assertion Result listener help us to understand assertion results.

### Size Assertion

This verifies size of response. It has three parameter **Apply To**, **Response Size Field to Test**, **Size to Assert.**

### XML Assertion

It validates the response only if it is in XML format & XML is well formness (like all tags are closed). It helps to find out faulty xml.

### XML Schema Assertion

This validates XML Schema.

### HTML Assertion

It checks for correctness of HTML document, if there are any tag missing.

### Response Assertion

### XPath Assertion

Validates xpath

### Compare Assertion

It compares result from two different server. To see result of this assertion, special listener **Compare Assertion Visualizer**

## Chapter 10 – Config Elements

It is used to setup default variables for later use by sampler. According to priority of execution they are executed first in the scope they are found (before any sampler in the same scope).

### Counter

Maximum value you can pass in counter is 2 ^ 63 -1

### Random Variable

### FTP Request Default

FTP Request is used to exchange files, either to retrieve the file or store a file on a server from local.

### Login Config Element

It contains the login details & other sampler will fetch login credential from this config element. This will be effective on request which are in same scope.

### HTTP Request Default

### Java Request Default

### User Defined Variables

User defined variables will be processed at the start of test.

### CSV Data Set Config

### HTTP Authorization Manager

### HTTP Cookie manager

## Chapter 20 – JMeter Inbuilt Functions

Function syntax in JMeter ${\_\_funcName(v1,v2,v3)}

**${\_\_time(YMDHMS)}** – Append this function in request name, this will add timestamp when request was executed.

**${\_\_Random(2000,5000,refVar)} –** to set random value in constant timer. Add this function in **constant timer -> Thread delay (in milliseconds).** refVar variable contains value generated by this function

Analyzing the critical business scenarios

Analyzing the service level agreement objectives

Developing the test scripts

Enhancing the scripts by creating/implementing

Parameterization

Checkpoints

Transactions

Reusable functions

Correlations

Actions/Blocks

Iterations

Pacing

Think Time

Designing workload model

Test Execution

Test Monitoring

Analyzing the test results

Preparing the test report

## Pointers

https://github.com/uttesh/JmeterAPISample/tree/master/lib

https://github.com/sayems/JMeter

http://uttesh.blogspot.in/2015/04/jmeter-load-testing-by-code-jmeter-api.html

http://www.agileload.com

https://www.blazemeter.com

https://www.blazemeter.com/blog/how-analyze-results-load-test-using-blazemeter-0

http://www.performancetestingfun.com/bottlenecks/

https://www.soasta.com/blog/

http://www.radview.com/

https://en.wikipedia.org/wiki/Arithmetic\_mean

https://en.wikipedia.org/wiki/Skewed\_distribution

https://en.wikipedia.org/wiki/Median

http://www.skillsyouneed.com/num/averages.html

**Rajendra Singh Some more links...**

http://www.quotium.com/performance/90-percentile-response-time/

http://www.loadtestingtool.com/blog/wapt-usage/how-to-analyze-load-test-report-response-time/

http://www.performancetestingfun.com/bottleneckanalysis/

http://sqa.stackexchange.com/questions/10090/load-testing-result-analysis-please-help-understand-a-few-keywords

https://www.thoughtworks.com/insights/blog/performance-testing-nutshell

http://topobaseinsiders.typepad.com/the\_topobase\_insiders/2007/09/oracle-database.html

**Open source performance monitoring tools**

\* http://www.slideshare.net/adrianco

\* http://www.strangeloopnetworks.com/blog/

\* http://www.perfdynamics.com/

\* http://www.igvita.com/

\* http://highscalability.com/

\* http://velocityconf.com/

\* http://blogs.impetus.com/test\_engineering/performance\_engineering/J2EEPerformanceTestingAndBottleneckIdentification.do

http://www.soasta.com/products/mpulse/

<https://www.blazemeter.com/blog/how-monitor-your-server-health-performance-during-jmeter-load-test>

## Performance Testing Strategy

* Determine the user load that will use the application during the peak time.
* We should gather these requirements from the customer, product owner or the business analysts.
* We should have scenario where we will have a set of virtual users who will cover the maximum load and the average load.
* We should design scenarios where we hit the system with a maximum load of users once and also a scenarios where will hit the system slowly as a group.
* We should also have a baseline established when we are going test against the newly added features.
* We should have a procedure to run a test against existing code (baseline) and then run the same test against the newly added code where we are able to measure against.
* performance test environment which is similar to the production environment. Otherwise we would not yield correct performance test results. The hardware configurations, database servers and the user load should be similar to the production site.
* We should perform the performance test near to the area where the application being deployed. If we do the performance test remotely the network latency will affect our performance test results.
* As a server side performance testing. We should note that we should performance test the web services and it the load to the web services and see whether they will work with the maximum load specified.
* We should also have access to the applications server hardware as we may have to obtain the information of server's processor, memory, databases and also OS processes that are being run, when we are doing performance testing. We should keep a close eye on these parameters as the utilization rate will increase ones there is high load of users.
* From database side, We should look at the utilization of stored procedures, how many stored procedures are executed and whether there are unwanted database calls created. This DB calls are also an underlying reason for performance issues in many of the web solutions.
* We should also be mindful to create end to end performance scenarios rather testing each feature separately, as testing end to end gives more realistic view.
* Tools - JMeter, Neoload

### Good Resources about JSON

<http://www.json.org/>

<https://en.wikipedia.org/wiki/JSON>

<https://www.w3schools.com/js/js_json_intro.asp>