API Testing

A guide on how to start testing an API

Date: 17 August 2015

REVISION HISTORY

Date	Version	Description	Author
07/28/2015	2.0	- Changed content and structure of the original document v1.0. - Explained approaches for different testing types in the main document. - Added Appendix A and B (Basic knowledge, Tools and Examples)	Claudia Coca, Karime Salomon, Edwin Taquichiri, Jafeth Garcia, Rina Espinoza, Eynar Pari, Josue Mendoza, Raul Cabero, Carlos Gonzales, Wilge Vargas, Jimmy Vargas
08/17/2015	2.0	- Added example of performance testing with JMeter	Eynar Pari

TABLE OF CONTENTS

TESTING METHODOLOGY

These are the regular phases that we need to follow when we start testing an API. Consider that this can vary according to the strategy you want to follow (check the Testing Strategies section).

1. PHASE 0 - SMOKE TESTS:

To make sure everything has been deployed and started correctly and identify if any part is not working or is missing, call every method once and get any response.

2. PHASE 1 - ACCEPTANCE TESTS:

a. Make CRUD (Create – Read – Update – Delete) tests to verify that objects are being written correctly in the database (Happy path).
b. Audit services and methods to make sure they have initial documentation.

3. PHASE 2 - FUNCTIONAL/NON-FUNCTIONAL TESTS:

a. Exercise each API method with a relevant variety of inputs for both positive and negative tests. Combinatorial tests (all-pairs), domain test, boundary and Equivalent Class Partitioning techniques would be used to determine what inputs and combination of should be tested.
b. Test the non-stateless API calls based on all the call combinations (dependencies between calls, etc.). State Based testing technique would be used for this kind of testing.
c. API end-to-end or Scenario Based tests. Behavioral Driven Development (BDD) would be applied to translate business scenarios to API tests.
d. Performance / Scalability testing would be performed, identifying thresholds for specific load of work.
e. Concurrency Testing, Internationalization Testing – I18N (calls using strings from uncommon char-sets as parameters), etc.
f. Time-Sensitive Data. Time-sensitive tests are a special subcategory of scenario based testing, and deals with verifying functionality depending on exact time or date.
g. Security tests would be done making sure access and permissions are accurate.
h. Documentation tests would be done.

4. PHASE 3 - CLIENT TESTS:

a. Clients would be developed in key languages (based on the expectations of use).
b. Document common flows using different calls would be part of the tests.

OVERVIEW ON SOME API TESTING TECHNIQUES

1. SMOKE TESTING

i. *DEFINITION:*

- End point is available
- It should return an status code different than 5xx (500 Internal Server Error, 503 Service Unavailable, etc.), but no need to validate the body response

ii. *EXAMPLE:*

Here some examples of what you should NOT receive as response, and they will make the smoke test fail:

- Response from the server 1 (*502 - Bad Gateway*):

```
HTTP/1.1 502 Fiddler - Connection Failed
Date: Fri, 10 Jul 2015 14:35:06 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Cache-Control: no-cache, must-revalidate
Timestamp: 10:35:06.468

[Fiddler] The connection to '172.21.3.43' failed. <br />Error: ConnectionRefused (0x274d). <br />System.Net.Sockets.SocketException No connection could be made because the target machine actively refused it xxx.xx.xx.xx:9090
```

- Response from the server 2 (503 - Service Unavailable):

```
HTTP/1.1 503 Service Temporarily Unavailable
Date: Fri, 10 Jul 2015 14:38:27 GMT
Content-Length: 400
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>503 Service Temporarily Unavailable</title>
</head><body>
<h1>Service Temporarily Unavailable</h1>
<p>The server is temporarily unable to service your
request due to maintenance downtime or capacity
problems. Please try again later.</p>
<hr>
<address>Apache/2.2.15 (CentOS) Server at 172.21.3.43 Port 80</address>
</body></html>
```

2. ACCEPTANCE TESTING

i. *DEFINITION:*

- It's based on the API Reference (Correct inputs, return expected outputs)

ii. *EXAMPLE:*

Soap Method	Param 1	Expected Output
getUserByID	10	<id>10</id> <name>Username</name> <lastname>Last</lastname> <account_id>ABC123</account_id>
getUserByName	Username	<id>10</id>

		<name>Username</name> <lastname>Last</lastname> <account_id>ABC123</account_id>

Rest Resource	Param 1	Expected Output
GET /users		*200 OK* <user> <id>10</id> <name>Username</name> <lastname>Last</lastname> <account_id>ABC123</account_id> </user> <user> <id>20</id> <name>Username2</name> <lastname>Last2</lastname> <account_id>XYZ789</account_id> </user>
GET /user/:id	20	*200 OK* <id>20</id> <name>Username2</name> <lastname>Last2</lastname> <account_id>XYZ789</account_id>
POST /user	<name>Username3</name> <lastname>Last3</lastname> <account_id>MNO456</account_id>	*200 OK* <id>30</id> <name>Username3</name> <lastname>Last3</lastname> <account_id>MNO456</account_id>

3. Functional testing (Testing of Functionality)

a. Exploratory testing

As its name implies, exploratory testing is about exploring, finding out about the software, what it does, what it doesn't do, what works and what doesn't work. The tester is constantly making decisions about what to test next and where to spend the (limited) time. This is an approach that is most useful when there are no or poor specifications and when time is severely limited.

Exploratory testing is a hands-on approach in which testers are involved in minimum planning and maximum test execution.

The planning involves the creation of a test charter, a short declaration of the scope of a short (1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used.

For more information, please review http://istqbexamcertification.com/what-is-exploratory-testing-in-software-testing/

b. Negative tests

- The purpose of any test is to try to break the application. A more specific way of testing is to use negative tests to bring out error messages. Error message should tell you exactly what went wrong: “a user must be logged in first”, or “missing amount for a deposit”. Obviously the errors should match the actual conditions.
- Error messages are especially important for public APIs; they help developers interface with your API. For internal APIs, it is quite common that the error messages will only return a code, and the list of codes is shared between different teams (server developers, UI developers, and testers).
- Errors therefore become a key tool providing context and visibility into how to use an API.
- Errors should not provide any application information i.e. database, table and columns name, class names in errors returned in the response
- Here some basic tests:
  A. Different Data Types
     - Check what happens if you send a String, and the method only receive an Integer.
  B. Number of parameters
     - If the method receives 2 parameters, try to send more (3, 4, etc) and less (1, none).
  C. Send null or empty values in required fields
  D. Send invalid/incorrect ID of the initialization predecessor object.
     - Example: the "initialization predecessor" is CountryUID for User object and in body request an CountryUID that does not exit is sent in POST method of the User object, then an error 4XX should be returned with a correct message.
  E. Validate the field’s length.
     - Example: the "userName" field only allows 20 characters and in the body request of the POST method we use a userName with more than 20 characters
  F. Conditional/Combinatorial rules.
     - Example: that field 1 is not necessary if field 3 is sent in body request

c. Fault Tolerance tests

To learn how the application handles miscommunication between components. The expected is that users are aware of any reduce of functionality and that transactions are not lost

i. Definition:

- Validate if SUT can still be working after any fault or component miscommunication

ii. Env Setup:

- Shutdown/block database
- Shutdown/block service
- Disable network connection between components (UI, service or DB)
- Any combination of above

iii. What to validate:

- Explicit exception is not displayed, only a friendly message
- Critical data is not exposed (DB names, table names, IP address, accounts, etc)
- API can continue with transactions once connection is reestablished

Note: This testing is also executed in Non-Functional Testing (i.e. Performance, Stress, Security,…)

d. Scenario-Based tests (End-to-End testing)

When it comes to testing the API, the Scenario tests should cover all of the business logic. That way, the UI team is allowed to concentrate on the user aspects: presentation and usability.

Scenario-based tests should be directly tied to user stories, which were probably provided to you by the product owner or some business stakeholder.

i. Strategies to Create Good Scenarios:

- Enumerate possible users their actions and objectives
- Evaluate users with hacker's mindset and list possible scenarios of system abuse.
- List the system events and how does the system handle such requests.
- List benefits and create end-to-end tasks to check them.
- Read about similar systems and their behaviour.
- Studying complaints about competitor's products and their predecessor.

ii. Scenario Testing Risks:

- When the product is unstable, scenario testing becomes complicated.
- Scenario testing is not designed for test coverage.
- Scenario tests are often heavily documented and used time and again

Other definition of the end to end testing is: The End-to-end testing is a technique used to test whether the flow of an application right from start to finish is behaving as expected. The purpose of performing end-to-end testing is to identify system dependencies and to ensure that the data integrity is maintained between various system components and systems.

The entire application is tested for critical functionalities such as communicating with the other systems, interfaces, database, network, and other applications.

The test cases cover the customer scenarios.

For example: One application will be integrated to Salesforce, to perform this integration the Salesforce provides an Mobile Application and API that the application will be use synchronize the Data (pull and push), but also the application will use other tools to integrate other functionality (i.e. ETL) the QA team will use the “end to end” technique to create the test cases, then:

- The test cases will be focus in most common and important scenarios that user performs using the Mobile Application (inputs and outputs)
- Define, the rules and make sure the correct outputs are showed up in the Mobile Application
- The testing should not be focus in internal processes.

e. State-Model Based tests

A program moves from state to state. In a given state, some inputs are valid, and others are ignored or rejected. In response to a valid input, the program under test does something that it can do and does not attempt something that

it cannot do. In state-base testing, you walk the program through a large set of state transitions (state changes) and check the results carefully every time.

4. NON-FUNCTIONAL TESTING (TESTING OF CHARACTERISTICS)

a. PERFORMANCE TESTS

i. DEFINITION:

- Determine how fast some aspect of a system performs under a particular workload. The workload is similar that will use the customers in their environments
- Helps to determine if system meets performance criteria and it does not have bottlenecks
- It can compare two systems to find which performs better
- It is focused in: how fast and stable is the system?

ii. OBJECTIVE:

- Forecast in advance the performance and degradation problems on system resources, before "the application" goes to production and facilitate their correction.

iii. ENV SETUP:

- Put the SUT under a regular load (Network, Data)
- Setting the necessary tools to monitor the resources used by API

iv. WHAT TO MONITOR:

- Service Oriented - the indicators are: Availability and Response time
- Efficiency Oriented - the indicators are: Throughput and Utilization

v. ANALYSIS:

- Compare Response time to a base line.
- A base line can be taken running the same tests in a previous version or build of the product.
- If there is no base line, ask for expected max time of response to the Product Owner or Business Analyst.

b. LOAD TESTS

i. DEFINITION:

- Determine if the system can handle its expected number of transactions
- Help to identify bottlenecks in different monitored points in the SUT (DB, Application Servers, etc.)
- The application's behavior under a specific expected load, to determine a system's behavior under both normal and at peak conditions.

ii. OBJECTIVE:

- Define the maximum amount of work a system can handle "How much" without significant performance degradation.

More references: http://www.soapui.org/Getting-Started/load-testing.html

c. VOLUME TESTS

i. DEFINITION:

- Determine if the system can handle a large amount of data

ii. OBJECTIVE:

- The goal is to determine system performance, with increasing volumes of data in the database.

d. SPIKE TESTS

i. OBJECTIVE:

- The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load

e. ENDURANCE OR SOAK TESTS

i. DEFINITION:

- Determine if the system can sustain the continuous expected load during a significant period of time
- Memory utilization is monitored to detect potential leaks

ii. OBJECTIVE:

- The goal is discover how the system behaves under sustained use. That is, to ensure that the throughput and/or response times after some long period

iii. ENV SETUP:

- Generate a high load according to the desired expectations by:
  - making several calls of different methods
  - simulating more clients doing calls

iv. WHAT TO MONITOR:

- Values returned are still correct
- Run it for several Hrs, days or weeks
- Counters start to cross the threshold of a good performance
- Counters not freeing memory (memory leaks)

v. ANALYSIS:

- (Server side) If we have the counters already collected (memory, disk, network, processor), then we can analyze the projection.
- We can use the following calculations using a linear projection:

$$Y = a + b * X$$

a: intercept
b: partial regression coefficient
Y: independent variable (time)
X: dependent variable

$$a = \frac{\sum X^2 \times \sum Y - \sum X \times \sum XY}{n \times \sum X^2 - \left(\sum X\right)^2}$$

$$b = \frac{n \times \sum XY - \sum X \times \sum Y}{n \times \sum X^2 - \left(\sum X\right)^2}$$

Sample:

X (Time)	Y (Counter)	XY	X^2	Y^2
1	21.7	21.7	1	470.89
2	27.2	54.4	4	739.84
3	38.6	115.8	9	1489.96
4	54.1	216.4	16	2926.81
5	66.3	331.5	25	4395.69
6	74.7	448.2	36	5580.09
7	82	574	49	6724
8	94.4	755.2	64	8911.36
9	100.2	901.8	81	10040.04
10	107.2	1072	100	11491.84
55	**666.4**	**4491**	**385**	**52770.52**

Finding **a** and **b** (n=10)

$$a = \frac{(385 * 666.4) - (55 * 4491)}{(10 * 385) - (55)^2}$$

$$a = 11.5867$$

Finding **b:**

$$b = \frac{(10 * 4491) - (55 * 666.4)}{(10 * 385) - (55)^2}$$

$$b = 10.0097$$

Then the formula is:

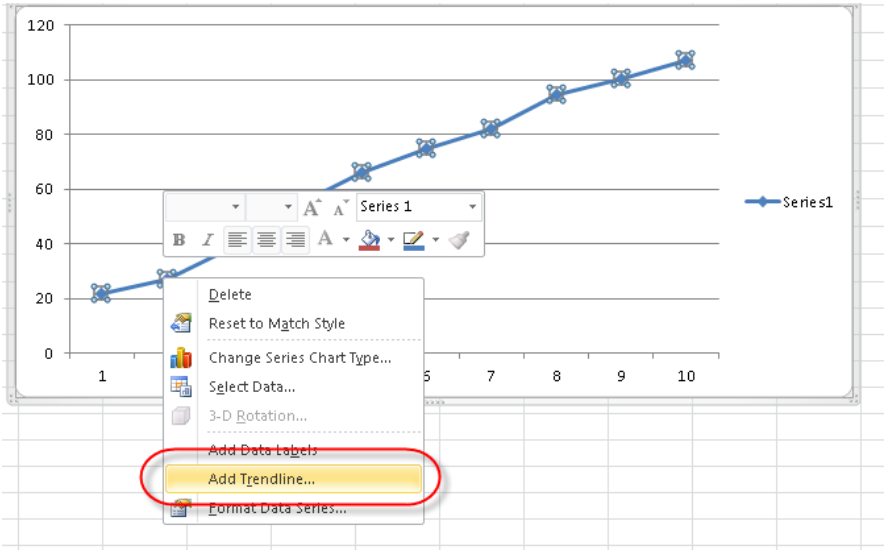
Y = 11.5867 + 10.0097 * X

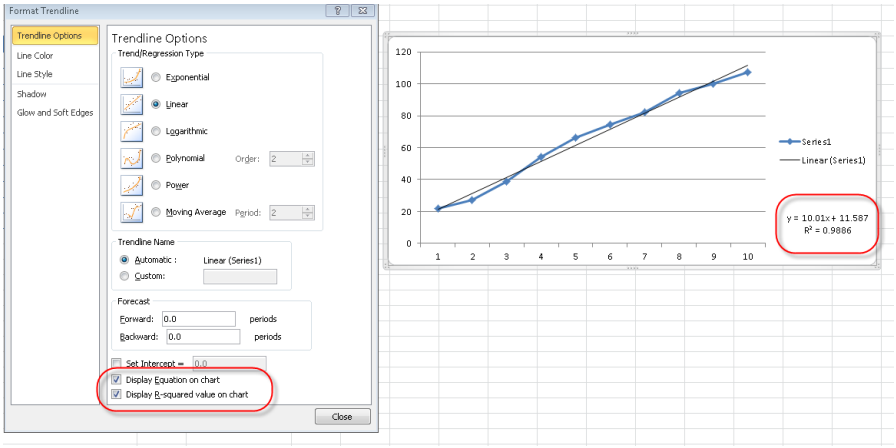
Then we can get the tendency if we replace the value of X:

Y = 100.5867 + 10.0097 * **11**

Y = 121.69

Note: In Excel we can get the tendency and the formula easily





Analyzing the slope: if we already have the formula of slope, we can analyze if there is memory leaks.

Formula:

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

Slope	Projection
Positive	ascending
Negative	descending
Zero	horizontally

We have to analyze the measure of counters and the slope to determinate if there is a memory leak in the future

i.e: if we have an slope positive (0.2) then if there is memory leak but if the measure of counter is in bits the memory leak will be a problem of low severity because It can collapse the memory, processor or other in a long time, but if the measure of counter is MB the memory leak is a problem of high severity because it can collapse the memory, processor or other in a few time.

f. SCALABILITY TESTS

i. DEFINITION:

- It measures the SUT capability to scale up in terms of any of its non-functional capability like load supported, the number of transactions, the data volume etc.

ii. ENV SETUP:

- Generate a high load according to the desired expectations by:
  - making several calls of different methods
  - simulating more clients doing calls
  - getting large amounts of data from the SUT

iii. WHAT TO MONITOR:

- Values returned are still correct
- Counters are under the desired parameters

g. STRESS TESTS

i. DEFINITION:

- It goes beyond the normal expected usage of the system (to see what would happen outside its design expectations), with respect to load or volume

- It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. It is a form of testing that is used to determine the stability of a given system, with emphasis on robustness, availability, and error handling under a heavy load

ii. OBJECTIVE:

- Ensure the software does not crash or it recovers in conditions of insufficient computational resources (such as memory or disk space).

iii. ENV SETUP:

- Generate a high load by:
  - making several calls of different methods
  - simulating more clients doing calls

iv. WHAT TO MONITOR:

- Values returned are still correct

h. CONCURRENCY TESTS

i. DEFINITION:

- Performed to identify the defects in an application when many users do actions at same time (i.e.: login to the application, create, update, delete , get records and others)
- It help us in identifying and measuring the problems in system parameters such as response time, throughput, locks/dead locks or any other issues associated with concurrency
- By using this technique the tester wants to learn how the product handles concurrent connections to the same objects. At the GUI level it is usually done by having multiple consoles opened trying to work on the same objects by different users

ii. ENV SETUP:

- EXAMPLE CASE1: Call a method to UPDATE a record in a loop, and call a method to DELETE the same record. No error should be displayed.
- RELATED TO USERS:
  - No need to play with different roles if all users do the same in the end point
  - Can use the same user account in all request (if API allows)
  - Can use different user account in each workflow (login > add > edit > delete > logout > repeat...)

iii. WHAT TO MONITOR:

- Values returned are corresponding to the correct user
- None of the calls should return an exception

i. CAPACITY TESTS

i. DEFINITION:

- Helps to determine how many users your application can handle before either performance or stability becomes unacceptable.

ii. ENV SETUP:

- Setup concurrency timers (how real users will interact with your SUT):
  - **Constant** – It pauses each thread for the same amount of time between requests.
  - **Gaussian Random** – This timer pauses each thread request for a random amount of time, with most of the time intervals occurring near a particular value. The total delay is the sum of the Gaussian distributed value (with mean 0.0 and standard deviation 1.0) times the deviation value you specify, and the offset value
  - **Uniform Random** – This timer pauses each thread request for a random amount of time, with each time interval having the same probability of occurring. The total delay is the sum of the random value and the offset value
  - **Constant Throughput** – This timer introduces variable pauses, calculated to keep the total throughput (in terms of samples per minute) as close as possible to a give figure.
  - **Synchronizing** – The purpose of the SyncTimer is to block threads until X number of threads have been blocked, and then they are all released at once. A SyncTimer can thus create large instant loads at various points of the test plan.
  - **Poisson Random** – This timer pauses each thread request for a random amount of time, with most of the time intervals occurring near a particular value. The total delay is the sum of the Poisson distributed value, and the offset value
- Simulate requests to a particular method

iii. WHAT TO MONITOR:

- Values returned are still correct

iv. REPORT:

- Response time numbers: Min, Max & Avg

  *Note: **Response Time = Latency + Processing Time**, we have to take into account the processing time for the real time that the process works when it is in the server side.*
- Transaction failure rate
- Transaction execution percentage
- Error percentage
- Throughput
- Server utilization at different load levels and variance in the numbers

v. CONSIDERATIONS:

- We need to know the max number of transactions that the application can support with the current license (if this consideration is applicable)
- We need to create the users, if the application don't work with virtual users
- We need to manage variable concatenated with the thread number if the application uses "tokens" to avoid problems with the authentication.

j. SECURITY TESTS

vi. DEFINITION:

- Software quality, reliability and security are tightly coupled. Flaws in software can be exploited by intruders to open security holes. With the development of the Internet, software security problems are becoming even more severe.
- Many critical software applications and services have integrated security measures against malicious attacks. The purpose of security testing of these systems include identifying and removing software flaws that may potentially lead to security violations, and validating the effectiveness of security measures. Simulated security attacks can be performed to find vulnerabilities.
- Security tests are another special case of negative tests. These involve sending specially crafted inputs to the SUT in an attempt to try to bypass normal access restrictions, and trick the application into revealing information which should not be revealed. Security testing is a very specialized area which requires much more research and education than others. Some testing tools offer canned security tests as part of their feature set, allowing you to easily expose your SUT for some of the most common security attacks out there. However, there are security vulnerabilities, such as zero-day exploits, that no tool can reveal and which require a specialist in the area.

i. WHAT TO MONITOR:

- Only return information requested, and nothing more
- SSL certificates and privileges

ii. ENV SETUP:

- Here some Security testing techniques applied to API testing:
    - WSDL Scanning
        - Look for unnecessary functions or methods kept open in the Web services
        - It is important to protect WSDL file or provide limited access to it
    - Parameter Tampering
        - The attacker can submit different parameter patterns in order to crash the application
        - Verify if the Web Service application is performing any type of input validation
    - XPATH Injection
        - This attack uses parameter tampering
        - It can retrieve unauthorized information from DB
    - Recursive Payload
        - An attacker can create a deeply nested document, for example: 100000 levels
        - This may overload the processor when it parses the document
    - Oversized Payload
        - Attacker can send an extremely large payload
        - Degrade the performance of an XML parser
        - This may result a denial-of-service (DoS)
    - XML Flooding
        - The goal is to overload a WS by sending SOAP message request repeatedly
        - Requests might be valid but the XML processor may not be able to process excessive requests in a short period of time

k. TIME-SENSITIVE TESTS

i. DEFINITION:

- Time-sensitive tests are a special subcategory of scenario based testing, and deals with verifying functionality depending on exact time or date.
- In the simplest case your test environment may be connected to a third-party environment that does not operate 24 hours, but only during business hours. Worst case, you are working from a time zone that is directly opposite to when that environment operates, as is common with off-shoring. In this case your tests will have to run from a scheduled service during hours of operation for the third-party environment. You would preferably use a CI – Continuous Integration system.
- A scheduled environment requires significantly more logging.
- In a scheduled environment you would have to account for the much longer turnaround time for developing your tests: develop test one day, wait for results next day, adjust tests, repeat. A scheduled environment requires significantly more logging than what you would have normally. This is because you don't have the ability to see the test in real-time, and you can examine the test inputs and outputs only after the fact.
- A more complicated scenario is when you have to enter data one day, and get results another day. An example is a lottery system: you place bets one day, the lottery drawing could happen at the end of the week, and you get your results (wins and losses) the following day. To manage this, you will need to develop some additional logging mechanism to keep track of your test's activity one day, and have that log read by your test suite at a later time. In the example of a lottery, you will need to keep track of perhaps all the ticket IDs purchased, and when the right time comes check them all for winnings to be able to test redemption correctly.

l. INTERNATIONALIZATION TESTS

i. DEFINITION:

- I18N in API testing is a process in charge of making sure that the application's API (or functionality) will not be broken with the usage of foreign language text (char-sets)

ii. EXAMPLE:

- An example of I18N testing is validating the create, update, and read operations over an object which contains foreign language text; Delete operations will be most likely performed pointing to an ID so the foreign language text will not be involved.

m. CRUD TESTS

i. DEFINITION:

- It is a good idea to verify the data directly in the database after each operation. One way to use CRUD tests is to reveal problems in any caching mechanism between the API server and the database server. If your smoke tests reveal that separately create and read calls works, but create and read for the same object instance back-to-back does not, that is probably a caching problem.
- Another problem area that CRUD tests can reveal is concurrency issues. If you make 5 parallel withdrawal calls for all of your money, only one of the calls should succeed. If more than one succeeds, this is often a problem with incorrect locking mechanisms

n. DOCUMENTATION TESTS

i. WHAT TO MONITOR:

- Check API PRD if exists
- API reference

5. API CLIENT TESTS (WHEN PROVIDED BY JALA-CUSTOMER)

Some customers provide an API client that is a wrapper with extra functionality that helps to the final user to connect and make the calls in an easier way, for example in the case of XML-RPC and Soap APIs. They are usually oriented to a specific programming language.

This means that these API Clients need to be tested too.

It is usual to test mainly the extra functionality of the wrappers, for example: Connection, Authentication, etc., but you can always go further and test all the API calls using the API client provided by the customer. You will be creating your own client to check if the API client provided is working fine.

6. IF THERE ARE TOO MANY TCS TO TEST, USE ANY OF THESE TECHNIQUES:

a. COMBINATORIAL TESTS (ALL-PAIRS)

This technique is based on the observation that most faults are caused by interactions of at most t parameters. When t=2, the technique is called pair-wise testing or all-pairs. Different studies specify the level of t needed in order to get proper coverage per type of software, but for any kind of program, it is known that 6 is the greater t value needed. This technique reduces the need to test all possible combination of variables exhaustively therefore, the test suites are much smaller than exhaustive ones, but yet very effective while finding bugs. Combinatorial testing can be applied to both, input parameters and configuration parameters. This means that if you are to test an application that can run on 9 different OSs, 10 Browsers, 4 .Net Framework versions, 12 Office versions and 3 versions of your application; you would have to test 12,960 possible combinations (9x10x4x12x3). With combinatorial testing this is reduced to just 126 combinations. To build the test suites, use the ACT tool located in our file server. You will be asked to enter all variables and the possible values of each, as well as the t level required

b. EQUIVALENT CLASS PARTITIONING TESTS

An equivalence class is a set of values for a variable that you consider equivalent. Test cases are equivalent if you believe that (a) they all test the same thing; (b) if one of them catches a bug the others probably will too; and (c) if one of them doesn't catch a bug the others probably won't either. Once you've found an equivalence class, test only one or two of its members

c. DECISION TABLES

A decision table is a good way to deal with different combination inputs with their associated outputs and also called cause-effect table. Reason to call cause-effect table is an associated logical diagramming technique called 'cause-effect graphing that is basically use to derive the decision table.

Decision table testing is black box and white box test design technique to determine the test scenarios for complex business logic.

We can apply Equivalence Partitioning and Boundary Value Analysis techniques to only specific conditions or inputs. Although, if we have dissimilar inputs that result in different actions being taken or secondly we have a business rule to test that there are different combination of inputs which result in different actions. We use decision table to test these kinds of rules or logic.

For more information and examples, please review:

- http://istqbexamcertification.com/what-is-decision-table-in-software-testing/
- http://www.softwaretestingclass.com/what-is-decision-table-in-software-testing-with-example/
- https://www.youtube.com/watch?v=ED2iJXkdhCQ&safe=active

d. DOMAIN TESTS

A domain is a set that includes all possible values of a variable of a function. In domain testing, you identify the functions and the variables. For each variable, you partition its set of possible values into equivalence classes and pick a small number of representatives (typically boundary cases) form each class. The assumption of the method is that if you test with a few excellent representatives or a class, you'll find most or all of the bugs that could be found by testing every member of the class. Notice that in contrast to function testing, the primary element of interest is the variable rather than the function. Many variables are used by more than one function. The domain tester will analyze a variable and then, based on that analysis, run tests that involve this variable on each function with this variable as input or an output

e. BOUNDARY TESTS

An equivalence class is a set of values. If you can map then onto a number line, the boundary values are the smallest and largest members of the class. In boundary testing, you test these, and you also test the boundary values of nearby classes that are just smaller than the smallest member of the class you're testing and just larger than the largest member of the class you're testing. Consider a field that accepts integer values between 10 and 50. The boundary values of interest are 10 (smallest), 9 (smaller than the smallest), 50 (largest), and 51 (larger than the largest).

TESTING STRATEGIES

1. HOW TO PRIORITIZE THE TESTS?

a. Test the most used methods first. It can be based on a statistic summary of the most used methods of the Web Service retrieved from the logs of production.
b. Test methods related to a feature of the product, then move to the next feature. Consider checking which are the most critical features in the product.
c. Focus on methods that are not covered already by Unit tests.

2. HOW TO GET FASTER RESULTS TO CLIENTS?

a. Complete acceptance tests first in all services/methods, and then move to the next testing technique.

3. BUILDING FRAMEWORKS FOR API AUTOMATION

a. If you start with API manual testing
   - Consider using a tool that you can easily expand to automate a set of tests
   - Check with your client how often the tests need to be executed and show the value on having automated tests running

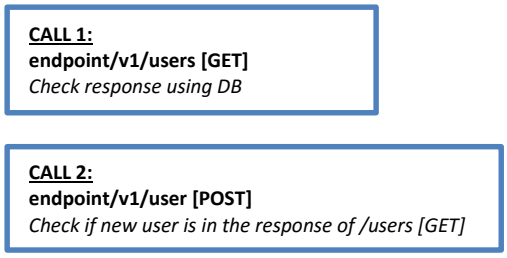
b. If you are building your own automation framework:
   - Model each web service as a class
   - Model each call as a method
   - Automated Script will use instances of these classes to create the steps of the TestCase
   - Always add the pain requests and responses in the framework logs for easy debugging
   - Use any unit test runner to organize and trigger your scripts
   - Consider having configuration files to choose:
     i. Endpoints
     ii. WSDL and WADL paths
     iii. Credentials
     iv. Web service version

4. WHAT'S BETTER, TEST BY TECHNIQUE OR BY API METHOD?

a. When development is just starting with the product, it's preferable to test by Method, as the development team is usually going to deliver one method as User Story.
b. If the Web Service is already there, probably you would like to test by technique, so you can cover most of the methods already available in the API.

BEST PRACTICES

a. Understanding the API organization
b. Automated tests are always better than doing the work manually every time, what cannot be tested by automation in API and must be tested manually.
c. Ideal/options:
   - You don't need to apply all techniques to test an API
   - Each TC should clean data in the SUT before running (use pre and post conditions to clean it)
   - Avoid dependencies between TCs
   - Environments:
     i. Make sure the environment/data is clean
     ii. Better to have one env for "automation only"
d. What do we need to validate in an API test?
   - Response Integrity
     i. Plain validation in response body/content (base line)
     ii. Compare retrieved values from the API with the SUT database
     iii. Use other API methods to validate other methods



This helps to avoid using static queries in multiple scripts when you are in front of a changing DB model (less maintenance).

- Header
    1. Status code
    2. Auth
    3. Content-type
        a. JSON/XML/OTHER

e. Results for non-functional tests: It's a good idea to include logs from execution, so development can see the root cause of the issue.

f. In concurrency, try to set up your environment as close as client's, as we need to consider the latency time.

g. These are some guidelines when you are doing a White box API testing:
- Client needs to share the source code
- Need to know how to deploy the API (if infrastructure allows it)
- Check code to find specific cases where the API could fail
- Test cases should be oriented to cover Use Cases of the product as a customer would do, not to check code

h. When testing different versions of APIs, keep in mind the following:
- Run a diff between different WSDL files to check what was changed (request parameters and response format)
- In case of REST API, you will need to check the API reference for changes (or release notes)
- Test the API services as if it were a new one
- Compare response times (performance)

i. In the case of testing secured APIs (HTTPS), don't forget about this:
- Check version of SSL
- Validate if data is really encrypted when using HTTPS by using a sniffer (WireShark)

j. An API call should be considered as a transaction and you must follow the following guidelines when it's tested (ACID), especially in Concurrency testing and Fault tolerance tests:
- **Atomicity** or "all or nothing". Either all changes are saved or nothing changed.
- **Consistency**. Data remains in consistent stage all the time
- **Isolation**. Other sessions don't see the changes until transaction is completed. Well, this is not always true and depends on the implementation.
- **Durability**. Transaction should survive and recover from the system failures

GLOSSARY

a. **API** – An application programming interface (API) is a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications.
b. **Web Service** – A Web Service is a type of API, almost always one that operates over HTTP (Though some, like SOAP, can use alternate transports, like SMTP). Defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network"
c. **Endpoint** – URL where you can find the web services and all their methods
d. **SUT** – System Under Testing
e. **AUT** – Application Under Testing

APPENDIX A

API Basics

1. HTTP PROTOCOL

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

HTTP functions as a request-response protocol in the client-server computing model. A web browser, for example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

Remember that in API Testing, you need to validate if the correct codes are returned based on the context and the actions. Some APIs can respond 200-OK for a failure and include the error in the body of the response. All validations depend on how the API was designed.

a. HTTP STATUS CODES

When a browser requests a service from a web server, an error might occur.

Status codes are divided in 5 categories:

1xx: Information
2xx: Successful
3xx: Redirection
4xx: Client Error
5xx: Server Error

Each category contains different HTTP status codes; here the most commonly used codes:

Code	Message	Description
100	Continue	The server has received the request headers, and the client should proceed to send the request body
200	OK	The request is OK (this is the standard response for successful HTTP requests)
201	Created	The request has been fulfilled, and a new resource is created
204	No Content	The request has been successfully processed, but is not returning any content
304	Not Modified	Indicates the requested page has not been modified since last requested
400	Bad Request	The request cannot be fulfilled due to bad syntax
401	Unauthorized	The request was a legal request, but the server is refusing to respond to it. For use when authentication is possible but has failed or not yet been provided
403	Forbidden	The request was a legal request, but the server is refusing to respond to it
404	Not Found	The requested page could not be found but may be available again in the future
409	Conflict	The request could not be completed because of a conflict in the request
500	Internal Server Error	A generic error message, given when no more specific message is suitable

You can find the full list here:

- http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

- http://www.w3schools.com/tags/ref_httpmessages.asp

b. HTTP METHODS (HTTP 1.1)

The HTTP method is supplied in the request line and specifies the operation that the client has requested. Standard is defined in RFC2616.

They are 8 methods in HTTP 1.1 specification:

Method	Description
OPTIONS	It represents a request for information about the communication options available on the request/response chain identified by the Request-URI.
GET	It means retrieve whatever information (in the form of an entity) is identified by the Request-URI.
HEAD	It is identical to GET except that the server MUST NOT return a message-body in the response.
POST	It is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
PUT	It requests that the enclosed entity be stored under the supplied Request-URI
DELETE	It requests that the origin server delete the resource identified by the Request-URI.
TRACE	It is used to invoke a remote, application-layer loop- back of the request message.
CONNECT	This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel.

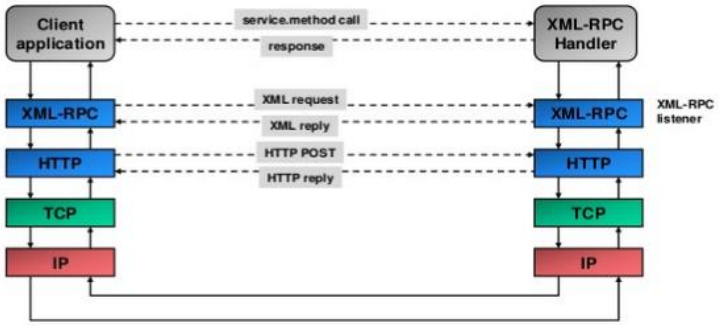
2. XML-RPC

a. DEFINITION

XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism.

b. ARCHITECTURE

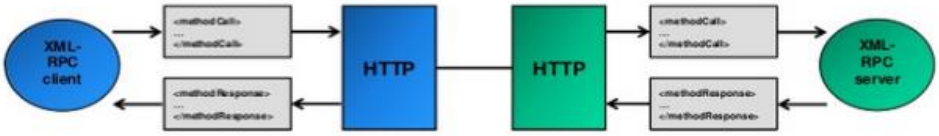
- The client application access to the server through URL (location where the server reside)
- The XML-RPC listener receives requests and passes this to the handler (user defined class servicing the request)



c. KEY POINTS

1. **Protocols and Techniques behind XML-RPC**

- XML. Formatting of the request and response and the arguments (wire protocol)
- RPC. Remote Call for procedures
- HTTP. Transport Protocol for the XML ("firewall-friendly")



2. **Base Types**

   XML-RPC has a very limited set of Base Types.

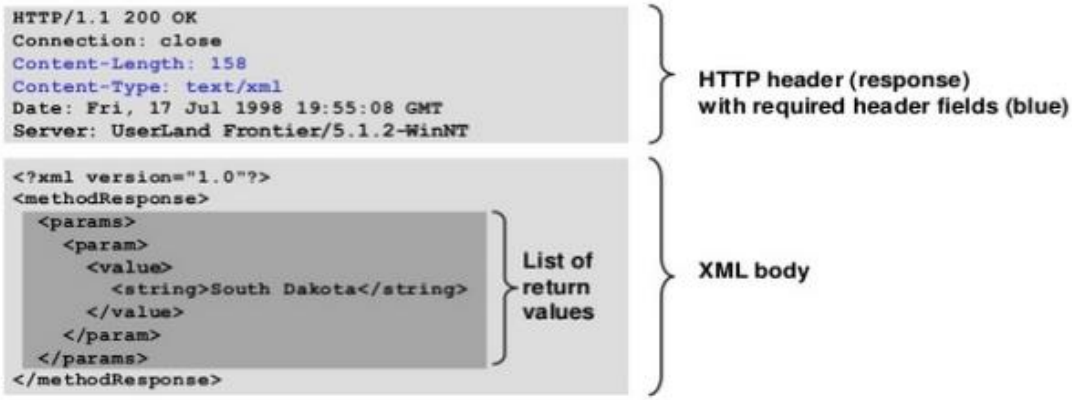
Type	Description	Example
<i4> or <int>	Four byte signed integer	-10
<boolean>	0 (false) or 1 (true)	1
<string>	ASCII string (this is by default)	Welcome
<double>	Decimal numbers	3.1415
<dateTime.iso8601>	Date/Time	19990105T14:05:00
<base64>	Base64 encode Binary	1uruie21urieur
<struct>	Associative array	<struct> <member> <name>foo</name> <value><i4>1</i4></value> </member> <member> <name>bar</name> <value><i4>2</i4></value> </member> </struct>
<array>	Array of values (no keys)	<array> <data> value><i4>1404</i4></value> <value><string>Something here</string></value> <value><i4>1</i4></value> </data> </array>

d. EXAMPLE

1. **Request** (example from http://xmlrpc.scripting.com/spec)

- The XML Body of http request contains a single method call getStateName
- The method does not need to be specified (service is indicated by string before the dot in method name element)

2. **Response** (example from http://xmlrpc.scripting.com/spec)



- The HTTP return response code is always "200 OK", even in case of an XML-RPC fault
- The response contains a single value (like a return argument from a local procedure call)

3. **Response with a Failure** (example from http://xmlrpc.scripting.com/spec)

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT
```

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member><name>faultCode</name><value><int>4</int></value></member>
        <member><name>faultString</name>
          <value><string>Too many parameters.</string>
          </value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

- Even in case of XML-RPC failure the HTTP response code is 200 OK
- The failure is indicated with <fault> element

3. SOAP

a. DEFINITION

SOAP stands for Simple Object Access Protocol.

It is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It uses XML Information Set for its message format, and relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

This XML-based protocol consists of three parts:

- an envelope, which defines the message structure and how to process it.
- a set of encoding rules for expressing instances of application-defined datatypes.
- a convention for representing procedure calls and responses.

b. ARCHITECTURE

The SOAP architecture consists of several layers of specifications for:

- Message format
- Message Exchange Patterns (MEP)
- Underlying transport protocol bindings
- Message processing models

- Protocol extensibility

SOAP evolved as a successor of XML-RPC, though it borrows its transport and interaction neutrality and the envelope/header/body from elsewhere (probably from WDDX).

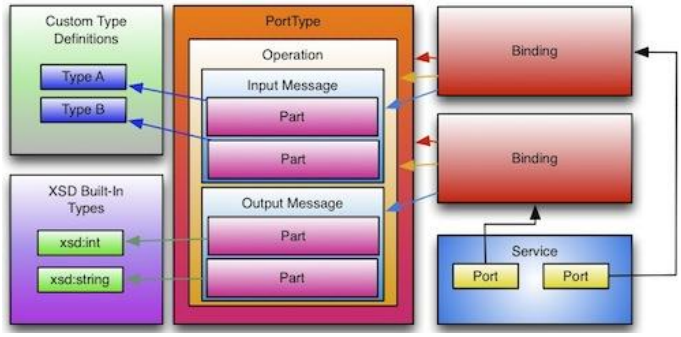
c. KEY POINTS

1. Descriptor (WSDL)

WSDL stands for Web Services Description Language.

It's a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

This is how the XML objects are organized and related:



2. SOAP Envelope

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

3. SOAP Header

The optional SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message.

If the Header element is present, it must be the first child element of the Envelope element.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

SOAP defines three attributes in the default namespace. These attributes are: *mustUnderstand*, actor, and *encodingStyle*.

The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

4. SOAP Body

The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.

Immediate child elements of the SOAP Body element may be namespace-qualified.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

The example above requests the price of apples. Note that the **m:GetPrice** and the **Item** elements above are application-specific elements. They are not a part of the SOAP namespace

5. SOAP Fault

The optional SOAP Fault element is used to indicate error messages.

If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault element has the following sub elements:

Sub Element	Description
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault

<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

The faultcode values defined below must be used in the faultcode element when describing faults:

Error	Description
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not proceed

6. SOAP Binding

Most SOAP implementations provide bindings for common transport protocols, such as HTTP or SMTP.

HTTP is synchronous and widely used. A SOAP HTTP request specifies at least two HTTP headers: Content-Type and Content-Length.

SMTP is asynchronous and is used in last resort or particular cases.

Content-Type

The Content-Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for the XML body of the request or response.

Syntax:

Content-Type: MIMEType; charset=character-encoding

Example:

POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8

Content-Length

The Content-Length header for a SOAP request and response specifies the number of bytes in the body of the request or response.

Syntax:

Content-Length: bytes

Example:

POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 250

d. EXAMPLE:

In the example below, a **GetStockPrice** request is sent to a server.

The request has a StockName parameter, and a Price parameter that will be returned in the response.

1. **SOAP Request**

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

2. **SOAP Response**

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

4. REST

a. Definition

Representational State Transfer (REST) is a software architecture style consisting of guidelines and best practices for creating scalable web services.

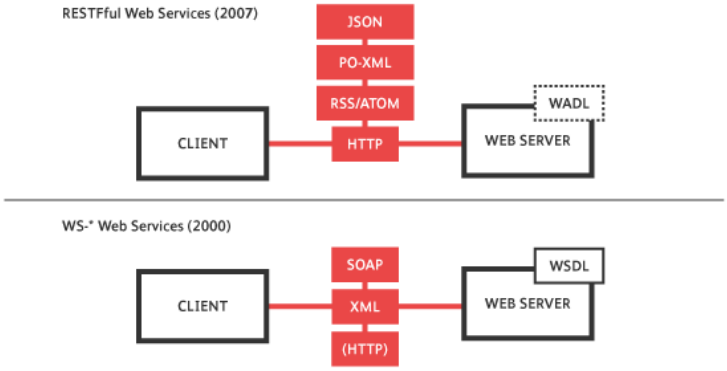
REST is a coordinated set of constraints applied to the design of components in a distributed hypermedia system that can lead to a more performant and maintainable architecture.

REST is often used in mobile applications, social networking Web sites, mashup tools, and automated business processes. The REST style emphasizes that interactions between clients and services is enhanced by having a limited number of operations (verbs). Flexibility is provided by assigning resources (nouns) their own unique Universal Resource Identifiers (URIs). Because each verb has a specific meaning (GET, POST, PUT and DELETE), REST avoids ambiguity.

REST and SOAP difference

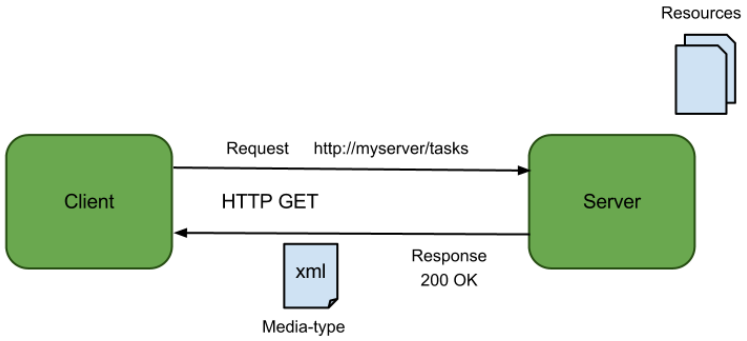
REST is an architectural style, unlike SOAP which is a standardized protocol. REST makes use of existing and widely adopted technologies, specifically HTTP, and does not create any new standards. It can structure data into XML, YAML, or any other machine readable format, but usually JSON – JavaScript Object Notation – is preferred.

REST follows the object oriented programming paradigm of noun-verb. REST is very data-driven, compared to SOAP, which is strongly function-driven.



b. Architecture Components

1. **Resources**, which are identified by logical URLs. Both state and functionality are represented using resources.
    - The logical URLs imply that the resources are universally addressable by other parts of the system.
    - Resources are the key element of a true RESTful design, as opposed to "methods" or "services" used in RPC and SOAP Web Services, respectively. You do not issue a "getProductName" and then a "getProductPrice" RPC calls in REST; rather, you view the product data as a resource -- and this resource should contain all the required information (or links to it).
2. A **web of resources**, meaning that a single resource should not be overwhelmingly large and contain too fine-grained details. Whenever relevant, a resource should contain links to additional information -- just as in web pages.
3. The system has a **client-server**, but of course one component's server can be another component's client.
4. There is no connection state; interaction is **stateless** (although the servers and resources can of course be stateful). Each new request should carry all the information required to complete it, and must not rely on previous interactions with the same client.
5. Resources should be **cachable** whenever possible (with an expiration date/time). The protocol must allow the server to explicitly specify which resources may be cached, and for how long.
    - Since HTTP is universally used as the REST protocol, the HTTP cache-control headers are used for this purpose.
    - Clients must respect the server's cache specification for each resource.
6. **Proxy servers** can be used as part of the architecture, to improve performance and scalability. Any standard HTTP proxy can be used.



c. Architecture Properties

1. **Performance** - component interactions can be the dominant factor in user-perceived performance and network efficiency
2. **Scalability** to support large Simplicity of interfaces
3. **Modifiability** of components to meet changing needs (even while the application is running)
4. **Visibility** of communication between components by service agents
5. **Portability** of components by moving program code with the data
6. **Reliability** is the resistance to failure at the system level in the presence of failures within components, connectors, or data numbers of components and interactions among components

d. Key Points

1. Descriptor (WADL)

The Web Application Description Language (WADL) is a machine-readable XML description of HTTP-based web applications (typically REST web services).

WADL models the resources provided by a service and the relationships between them.

WADL is intended to simplify the reuse of web services that are based on the existing HTTP architecture of the Web.

It is platform and language independent and aims to promote reuse of applications beyond the basic use in a web browser.

WADL is the REST equivalent of SOAP's Web Services Description Language (WSDL), which can also be used to describe REST web services.

Here is an example of a WADL document structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://research.sun.com/wadl" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ya="urn:yahoo:api" xmlns:yn="urn:yahoo:yn">
   <grammars>
      <include href="NewsSearchResponse.xsd" />
      <include href="NewsSearchError.xsd" />
   </grammars>
   <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
      <resource uri="newsSearch">
         <method href="#search" />
      </resource>
   </resources>
   <method name="GET" id="search">
      <request>
         <query_variable name="appid" type="xsd:string" required="true" />
         <query_variable name="query" type="xsd:string" required="true" />
         <query_variable name="type" type="xsd:string" />
         <query_variable name="results" type="xsd:int" />
         <query_variable name="start" type="xsd:int" />
         <query_variable name="sort" type="xsd:string" />
         <query_variable name="language" type="xsd:string" />
      </request>
      <response>
         <representation mediaType="application/xml" element="yn:ResultSet" />
         <fault id="SearchError" status="400" mediaType="application/xml" element="ya:Error" />
      </response>
   </method>
</application>
```

2. Authentication

Let's take a look at some of the existing practices:

- **API keys in URLs**: one example is the Finish National Gallery API (which I somehow stumbled upon). Here you are required to obtain an API key and put it into the URL in every request made to the API:

  http://kokoelmat.fng.fi/api/v2?apikey=********&q=A+III+2172

- **API keys in custom headers**: one example is from DocuSign. They require you to encode both your (developer) user name, password and API key in a JSON object and then send that in a custom header X-DocuSign-Authentication:

  GET /some-url
  X-DocuSign-Authentication: { "Username": "...", "Password": "...", "IntegratorKey": "..." }
  ... more ...

- **Signed URL/body parameters**: one example is last.fm which requires you to use your API key to obtain a session token and use that token later on to sign requests, either using URL parameters or in the body of the HTTP request.

- **HTTP Basic authentication**: GitHub supports authentication via the standard HTTP basic authentication mechanism where you supply username and password BASE64 encoded in the "Authorization" header.

- **OAuth1 and Oauth2**: PhotoBucket uses OAuth1 and BaseCamp uses Oauth2 (in addition to HTTP basic authentication).

e. REQUEST AND RESPONSE FORMAT:

Requests point to a particular Resource of an API Endpoint, and they use HTTP Methods to do the actions in the API, usually:

GET – Get an Item or a list of Items

POST – Add a new Item
PUT – Update and Item
DELETE – Delete an Item

As a response, they will return an HTTP Code number and the response in any supported format (JSON, XML, YAML, etc) according to the Content-Type configured.

1. REST Request

```
GET /myEndPoint/user
Accept: application/json+userdb
```

2. REST Response

```
200 OK
Content-Type: application/json+userdb

{
  "users": [
    {
      "id": 1,
      "name": "Emil",
      "country: "Sweden"
    },
    {
      "id": 2,
      "name": "Adam",
      "country: "Scotland"
    }
  ],
  "links": [
    {
      "href": "/user",
      "rel": "create",
      "method": "POST"
    }
  ]
}
```

APPENDIX B

Tools and Examples

1. MANUAL TESTING

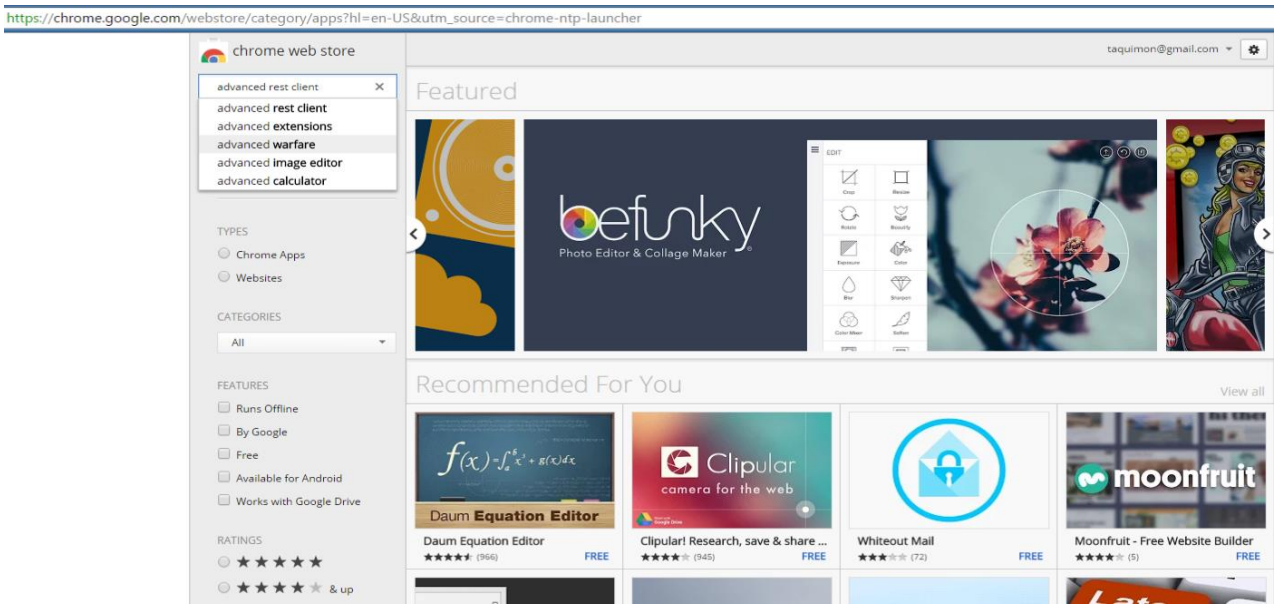
a. ADVANCED REST CLIENT

i. REQUIREMENTS

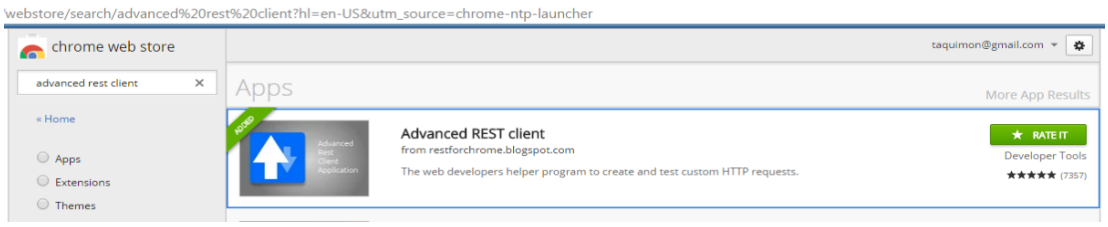
The API that we will use for this example is: http://todo.ly/ApiWiki/ (you need to have an account on http://todo.ly site)

ii. INSTALLING THE TOOL

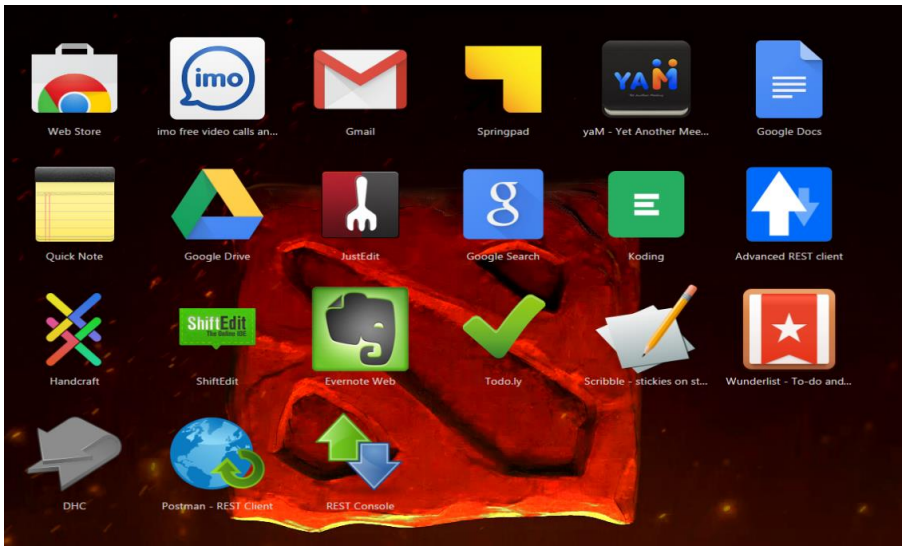
- Is a web program that help to create and test custom HTTP requests
- It can be installed from Chrome Web Store, you only need to search advanced rest client



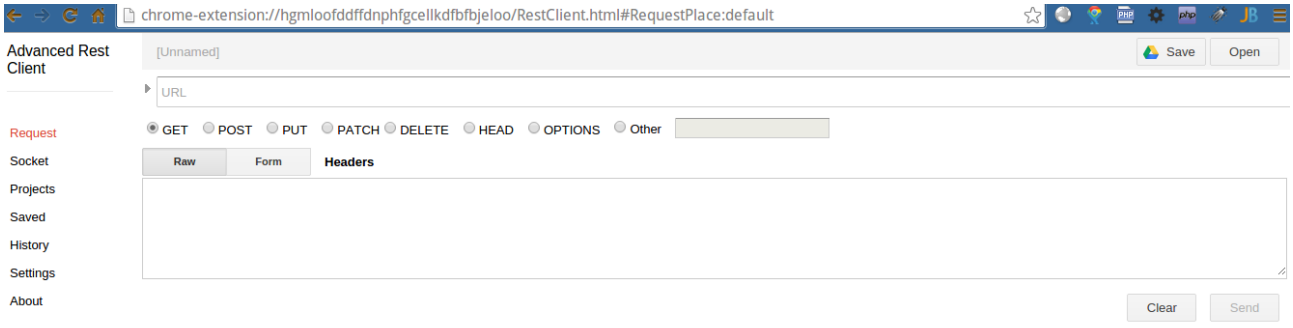
Then install the extension.



Once installed, you can open the extension by clicking on the app installed.



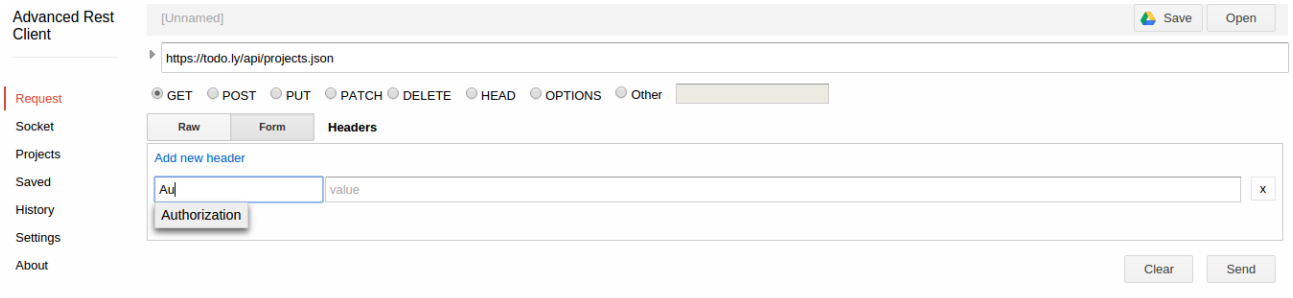
The image below should be displayed



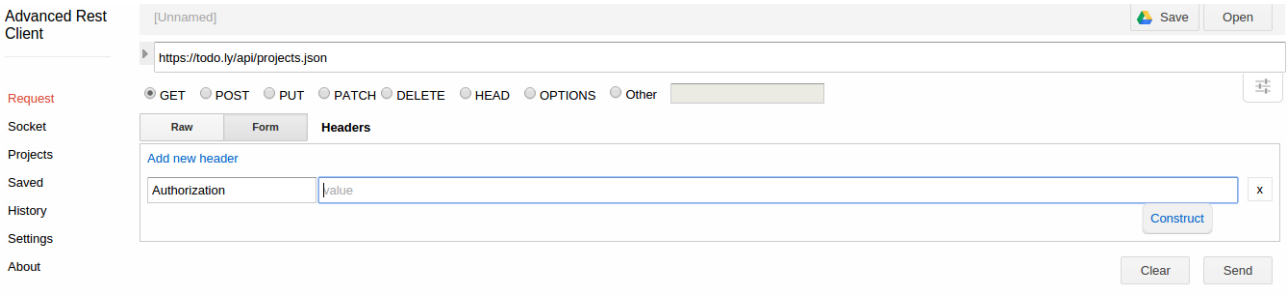
iii. EXAMPLE

First, you need to setup the credentials to use in the API call (username and password). We will add an Authorization.

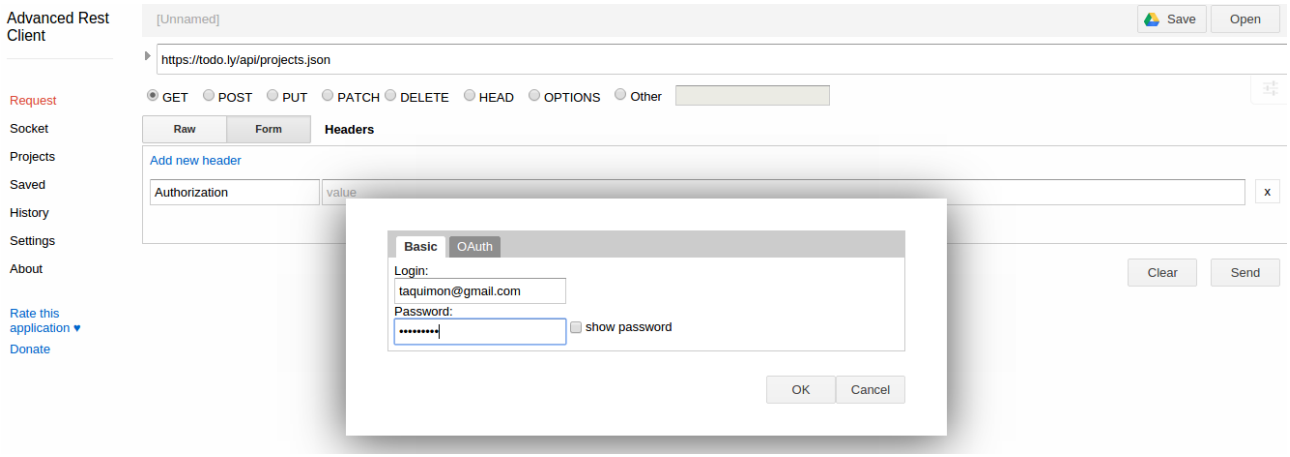
1. Enter the following: https://todo.ly/api/projects.json in URL textbox
2. Select GET option (this is selected by default).
3. Enter Authorization in **Headers** section.



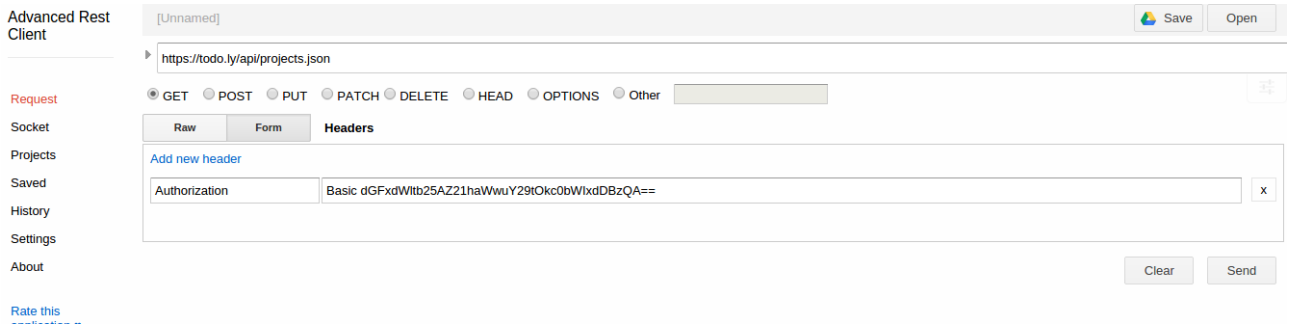
4. After this Construct button should be displayed.



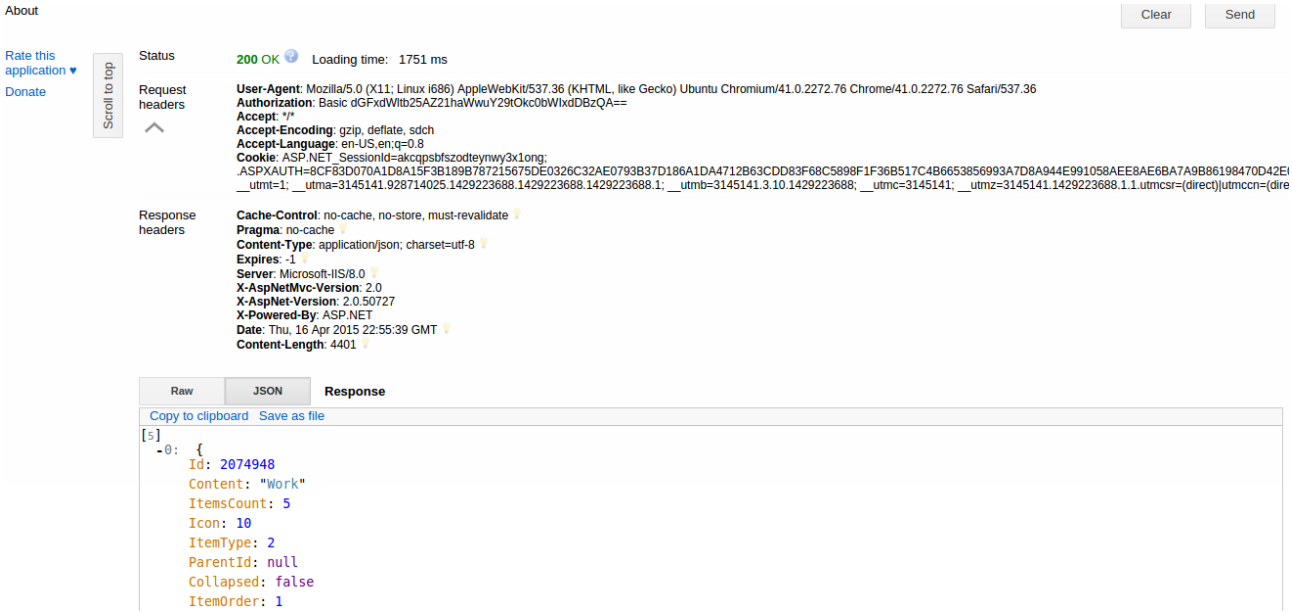
5. Now we can add a “Basic” or “OAuth” authorizations, in this case we choose “Basic” and fill with login and password.



6. Once filled the Authorization the request should look like the image below.

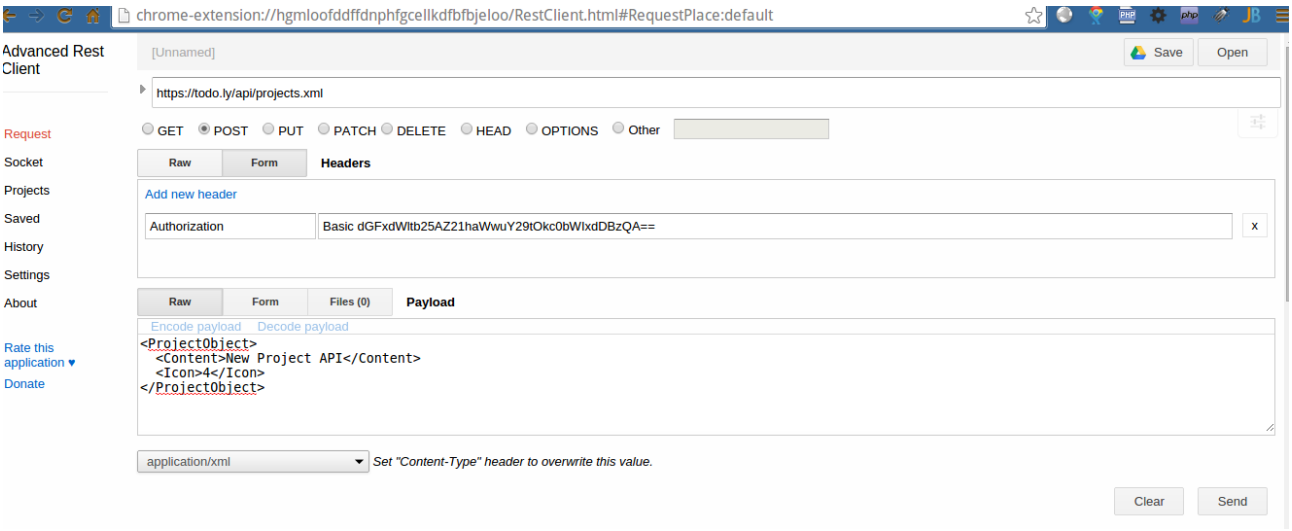


7. Next step is just click the “Send” button and the result should be displayed.

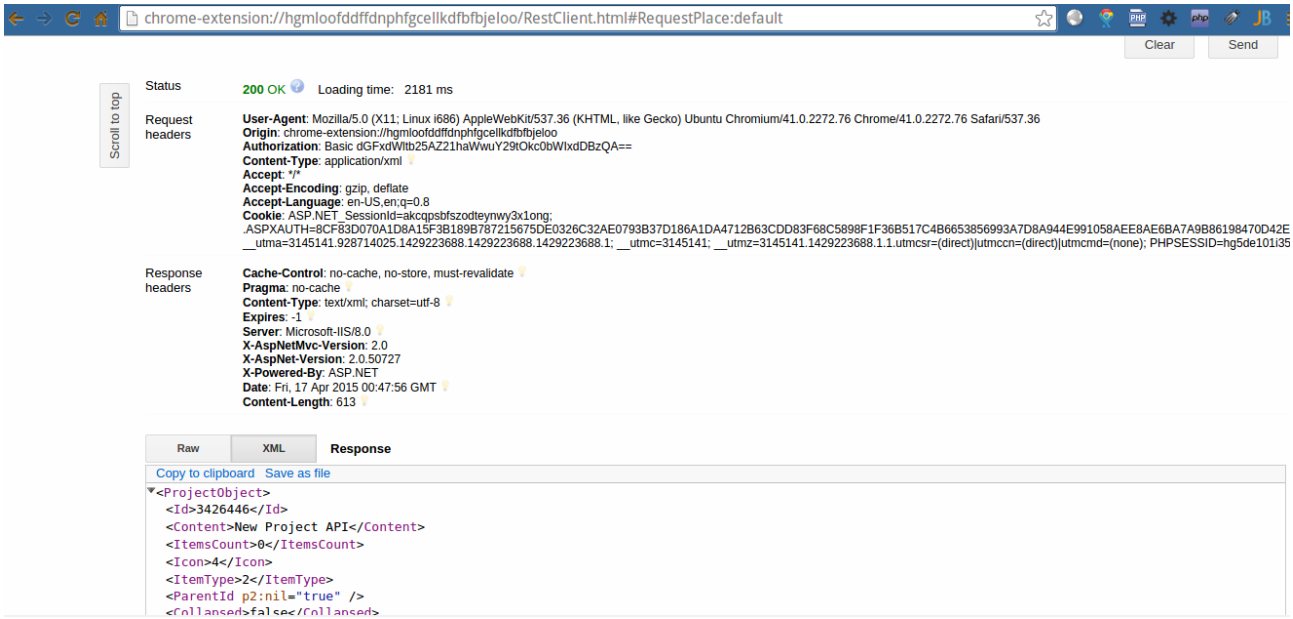


8. Now we going to do a POST example (Add a project) using the same authorization, the service URL in XML mode, and the parameters. You have to choose POST option and enter the XML object as parameters for the request:

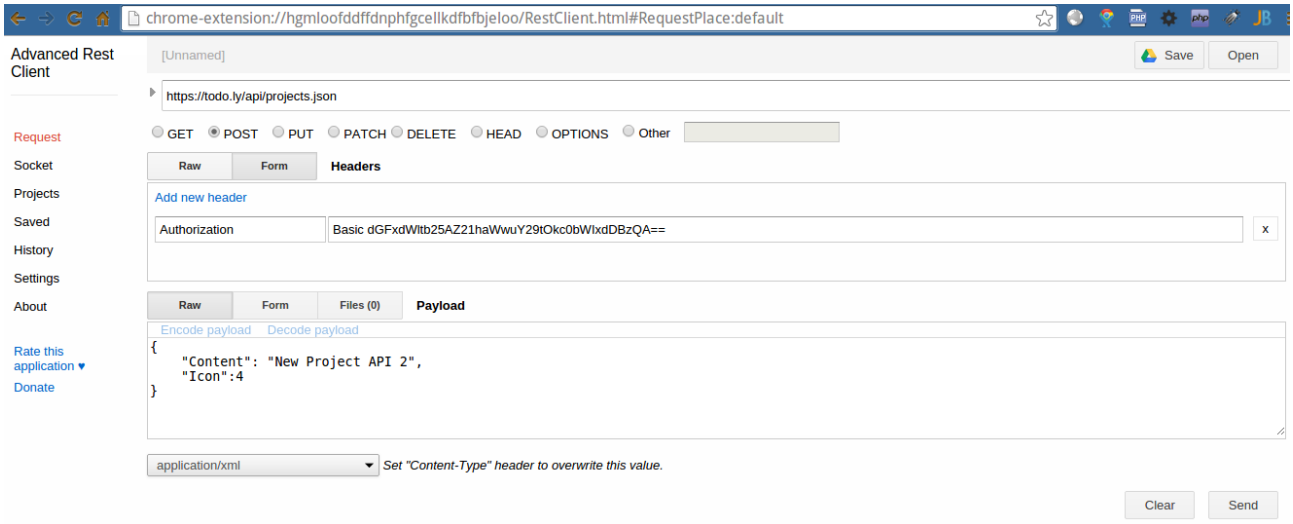
```
<ProjectObject>
  <Content>New Project API</Content>
  <Icon>4</Icon>
</ProjectObject>
```

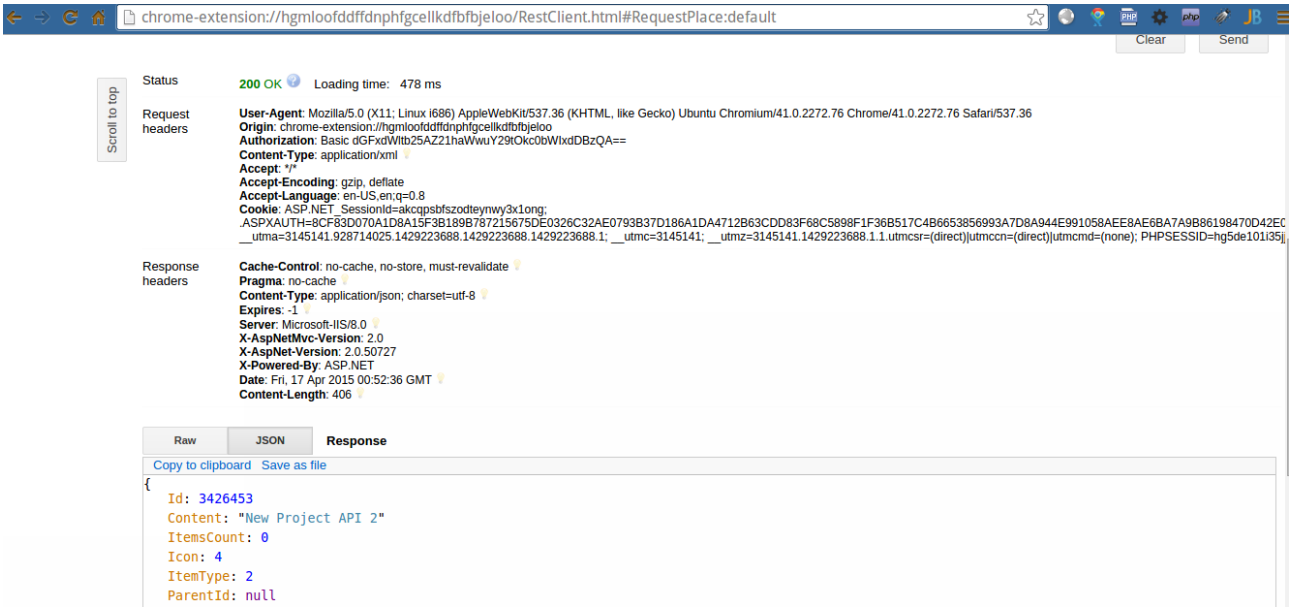


9. Finally click “Send” button and the response should be displayed.



10. Below is the same example, but is this case we are using JSON format for the parameters in the request.





b. POSTMAN

i. ABOUT THE TOOL

It is a chrome extension, with Postman, you can construct simple as well as complex requests quickly, save them for later use and analyze the responses sent by the API.
Using Postman you have the chance to interact with the API, sending different request (GET, POST, PUT, DELETE, etc) and receiving difference responses from them.

This is very useful tool because you can send different values, expected values, invalid values (empty, spaces, invalid character, extra-large, etc, etc) in order to verify if the service is giving a good response to this kind of request.

ii. CONFIGURATION

When begin using Postman, you need to consider the options described below:

a. **URL** - The endpoint to the API
b. **Method** - The http method that you are going to use
c. **Headers** - There are many type of headers, you need to define each one according the requirement of each service, the most important are:
   - Authentication - We can define authentication in the header, if the service is requiring some authentication every time that the service is called.
   - Content-Type - This header should give the type of data that will be received, for example if we are sending an XML or Text or JSON file, we need to set in the header which kind of data will be sent in order to have it recognized properly.

   Postman has many other options that you need to define if the service requires them.
d. **Request** - For the cases of POST, PUT and also DELETE, we are expecting to send some data to the service, so the service will process it. If we are sending this data in a specific format, we need to be sure to select it before send the request.

e. **Response** - It is the result that the service is giving you according the request send, you could receive a HTTP Code and also some kind of message or information in a specific format

iii. EXAMPLE

In order to give a better example we will use this service: https://todo.ly/api/.

Let's create a user:

a. Request:

URL: https://todo.ly/api/user.format

Format: json

HTTP Method: POST

Json Request:

```
{
  "UserObject": {
    "Email": "user@email.com",
    "FullName": "User_example",
    "Password": "P4ss"
  }
}
```

b. Response:

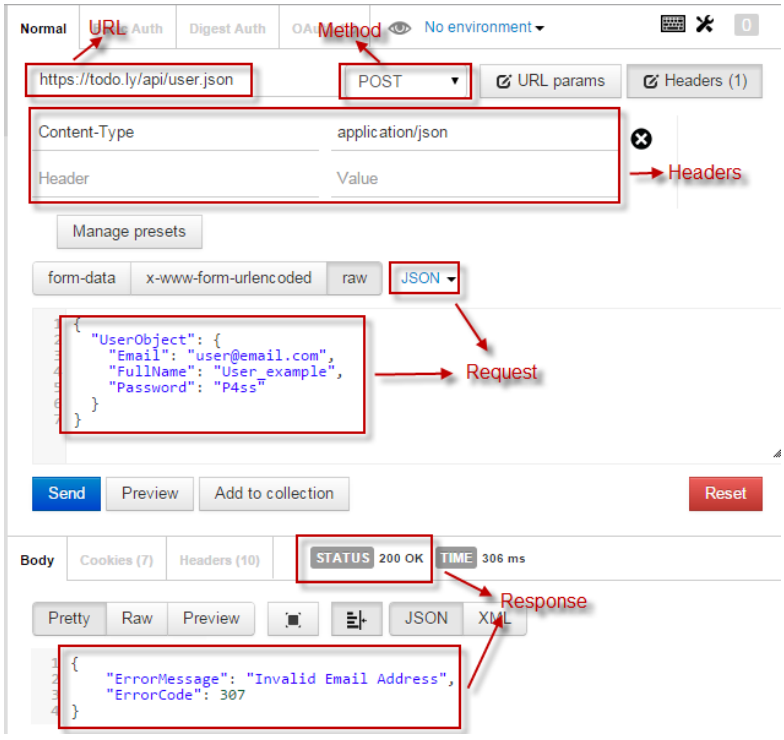
The response could be only an HTTP Code, or could be also some message according the case of the validation applied over the data sent.

HTTP Code: 200

Body:

```
{
    "ErrorMessage": "Invalid Email Address",
    "ErrorCode": 307
}
```

Configuring postman with this data will look like this:

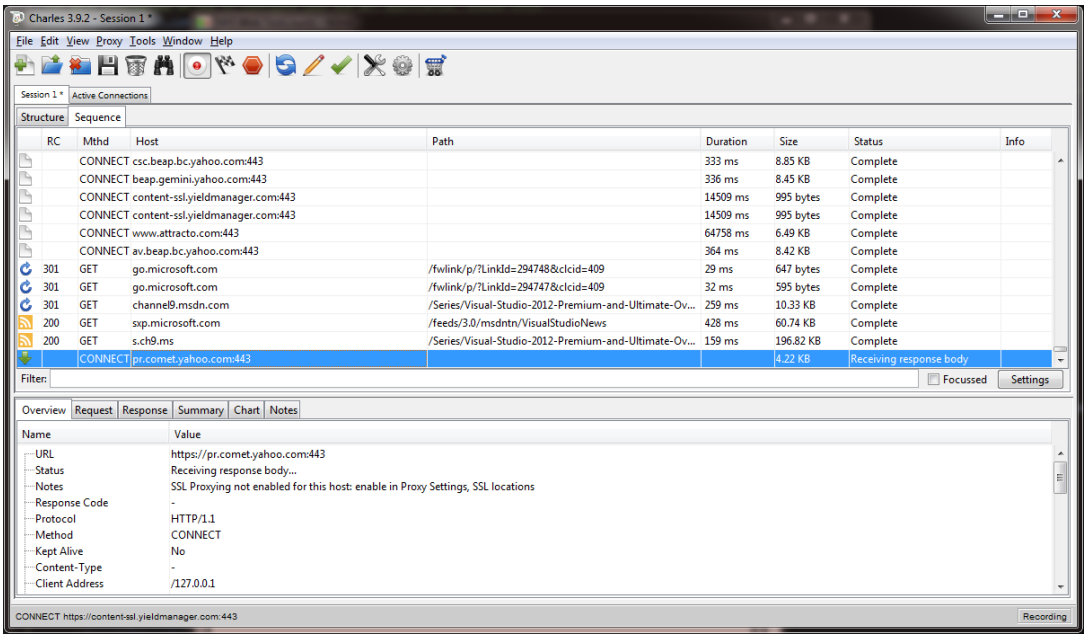


c. Charles Proxy

i. About The Tool

Charles is an HTTP proxy / HTTP monitor / Reverse Proxy that enables a developer to view all of the HTTP and SSL / HTTPS traffic between their machine and the Internet. This includes requests, responses and the HTTP headers (which contain the cookies and caching information).
It can be downloaded here: http://www.charlesproxy.com/download/

ii. Example



d. Fiddler

i. About The Tool

Fiddler is a free web debugging proxy tool for any browser, system or platform (http://www.telerik.com/fiddler). This tool can help you on:

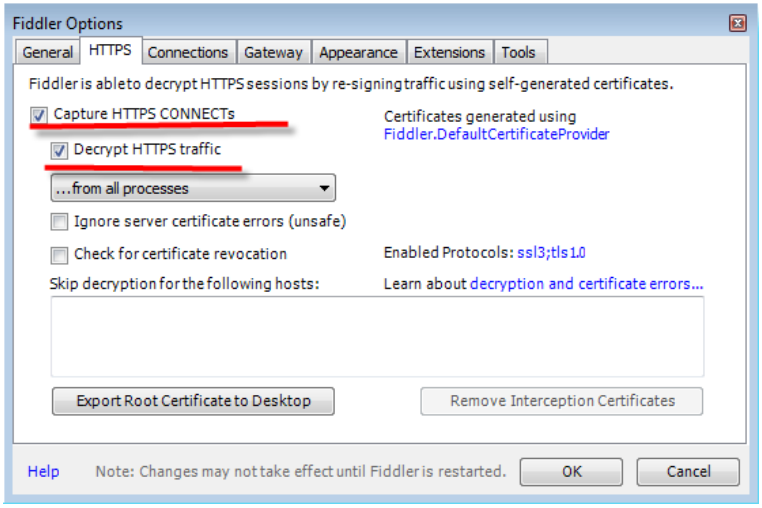
- Web Debugging
- Web Session Manipulation
- HTTP/HTTPS traffic recording
- Performance Testing
- Security Testing

For example you can use the Fiddler to:

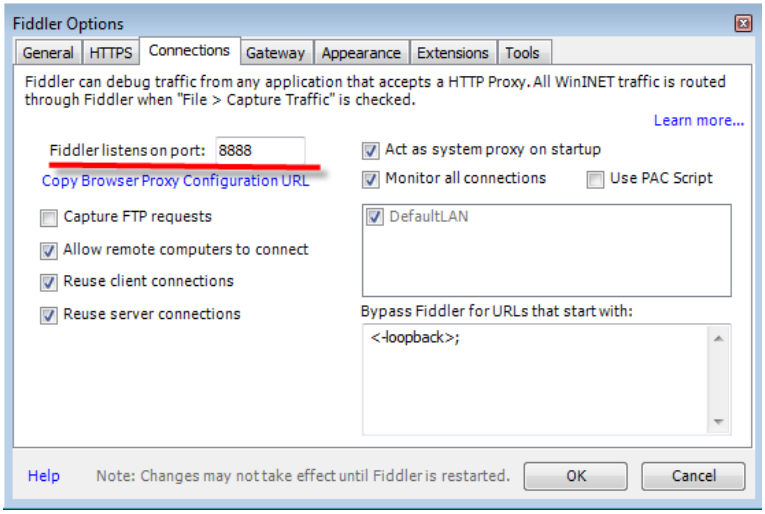
- Monitor, capture, debug and analyze all request and response sent by:
  - Two integrated projects
  - Review the requests and responses sent by client and server
- Help on QA testing, setting breakpoints, manipulate any HTTP(s) request or response. For example manipulate the status code sent in response.

ii. EXAMPLE

1. Go to Tools\Fiddler Options\
2. Select the "HTTPS" Tab
3. Check the Capture HTTPS CONNECTs and Decrypt HTTPS traffic check boxes



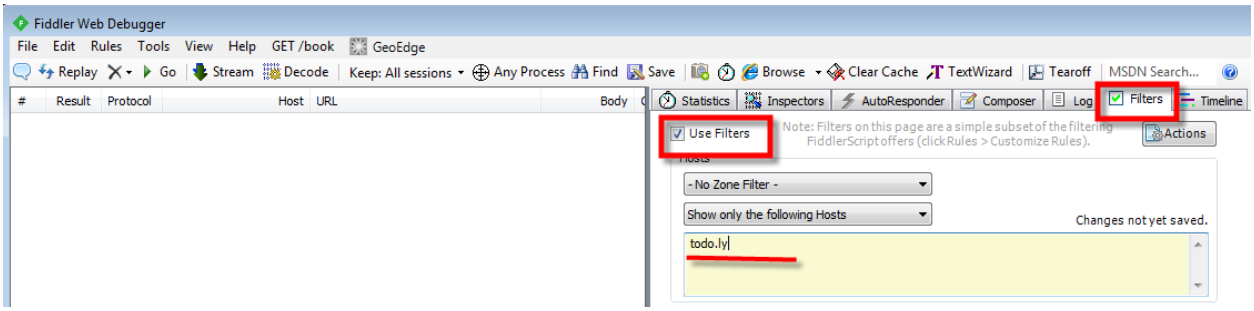
4. Select the "Connections" and review the port is setting properly tab



5. Press OK button in Fiddler Options dialog box
6. Filter the traffic that will be capture by Fiddler, selecting the Filters tab.
7. Check the "Use Filters" checkbox
8. Introduce the host name

***Notes**:*

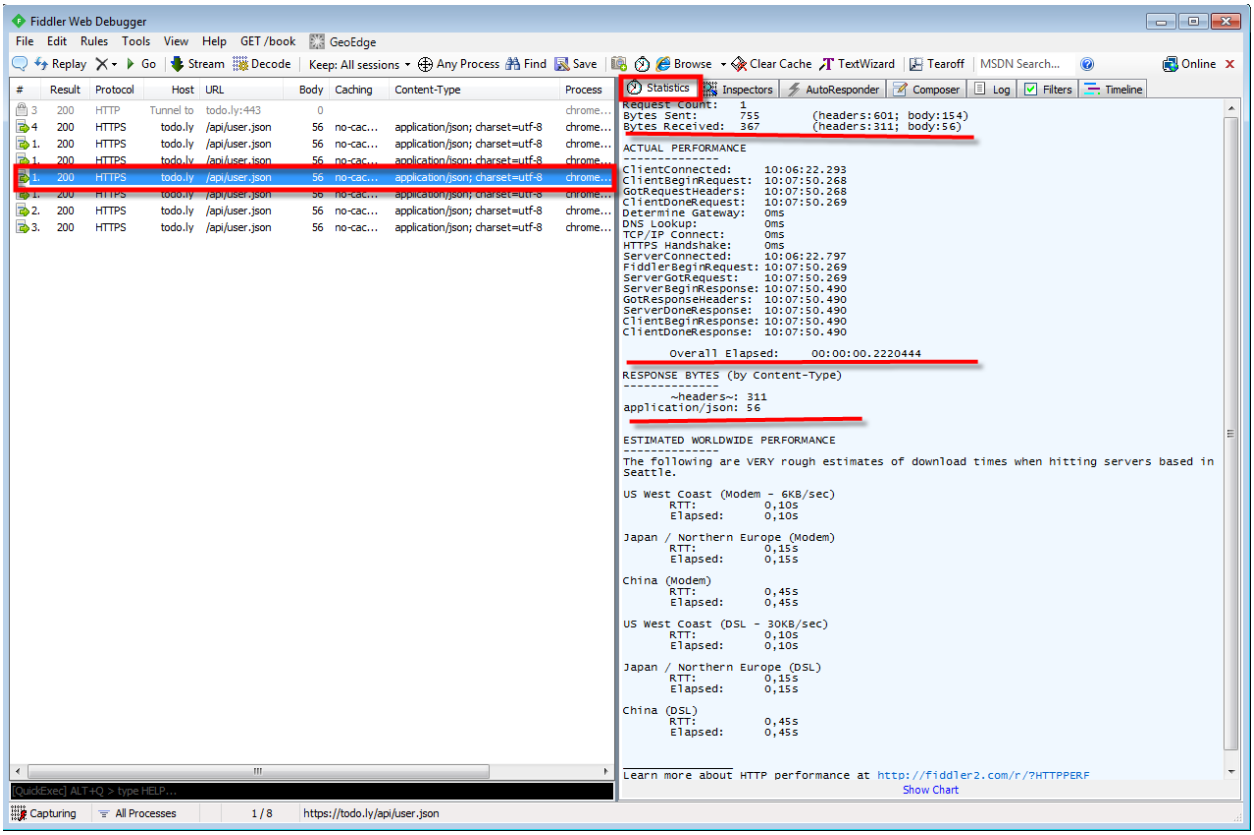
- *Multiple hosts can be introduced, those should be detached by ";" i.e. todo.ly; facebok.com*
- *If filters are not setting, all traffic will be captured i.e. if you have other web pages open*

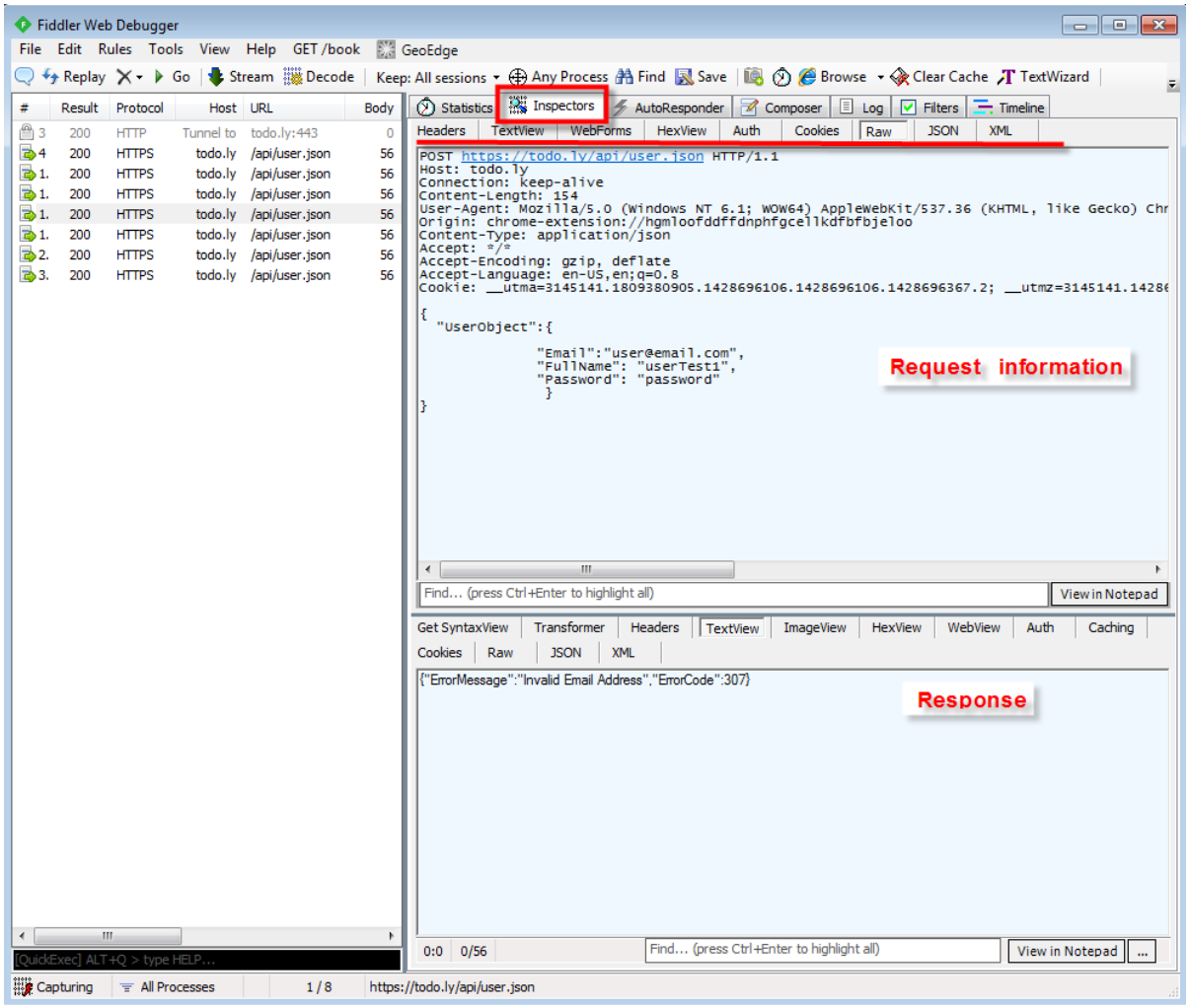


9. Perform operations in Client or sent request to API with some tool or other device
10. The sessions are capture in left panel of the Fiddler; in the right panel the information by each session selected is showed up in Sadistic, Inspectors, Logs … tabs

For example Statistics tab shows up all information about bytes sent, received in request/response, how much time did it take?…, and in Inspectors tab, in top panel all request information is available, the headers, request body in different formats

And in bottom panel all response information is showed up





e. CURL (PLAIN)

i. ABOUT THE TOOL

Curl is a command line tool and library for transferring data with URL syntax, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMTP, SMTPS, Telnet and TFTP. curl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, HTTP/2, cookies, user+password authentication (Basic, Plain, Digest, CRAM-MD5, NTLM, Negotiate and Kerberos), file transfer resume, proxy tunneling and more.

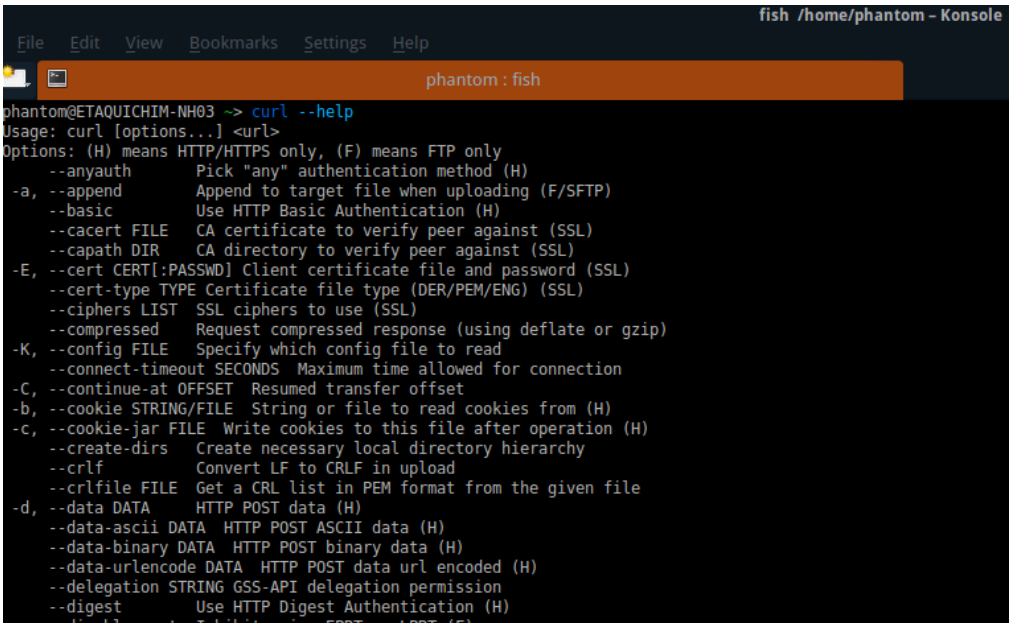
ii. INSTALLATION

- Linux Systems have the program installed by default
- Windows Systems need to install the program that can be downloaded from http://curl.haxx.se/download.html

You can verify if curl is functional by typing in a terminal the following command:

```
curl --help
```

The image below should be displayed. You can notice that there are a lot of options very useful that can help in your testing



iii. EXAMPLE

We will use todo.ly public API (https://todo.ly/api/user.xml). In order to use this API you should create an account on the site.

Now we are going to run some examples:

1. First, we will test the site without authentication. For this, please type the following in the terminal:

```
curl https://todo.ly/api/authentication/IsAuthenticated.xml
```

2. Now type

```
curl -u "username:password" https://todo.ly/api/authentication/IsAuthenticated.xml
```

Note that -u option is used to send user and password to server so this way we are calling API with authentication

3. The result should looks like image below:

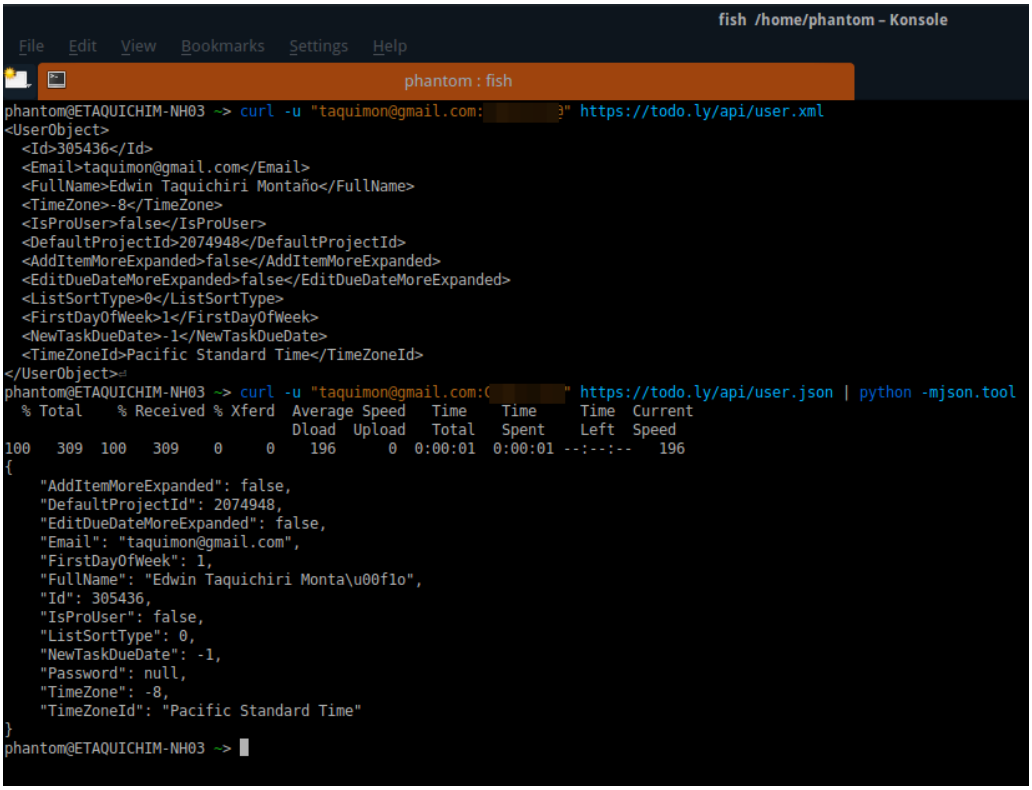
4. Now we are going to do a GET call. For this we only need to type in terminal (curl -u "username:passsword" url)

```
curl -u "username:password" https://todo.ly/api/user.xml
```

or

```
curl -u "username:password" https://todo.ly/api/user.json | python -mjson.tool
```

5. The line: "**| python -mjson.tool**" help us to prettify the output. This only works if you have installed python.
6. Below image shows both results xml and json.



7. Next example is related to POST calls for this we going to use 2 options

   - use the parameters in the same curl call
   - use an external file with the POST parameters

8. For first call we will add -d option and the POST parameters as string, type in terminal:

```
curl -u "username:password" -d '{ "Content": "New Project API 3", "Icon":4 }' https://todo.ly/api/projects.json | python -mjson.tool
```

```
phantom : x11vnc                                    phantom : fish
phantom@ETAQUICHIM-NH03:~$ fish
Welcome to fish, the friendly interactive shell
Type help for instructions on how to use fish
phantom@ETAQUICHIM-NH03 ~> curl -u "taquimon@gmail.com:[redacted]" -d '{    "Content": "New Project API 3",    "Icon":4 }' https://todo.ly/api/projects.json | python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   456  100   406  100    50    316     38  0:00:01  0:00:01 --:--:--   316
{
    "Children": [],
    "Collapsed": false,
    "Content": "New Project API 3",
    "Deleted": false,
    "Icon": 4,
    "Id": 3428975,
    "IsOwnProject": true,
    "IsProjectShared": false,
    "IsShareApproved": false,
    "ItemOrder": 17,
    "ItemType": 2,
    "ItemsCount": 0,
    "LastSyncedDateTime": "/Date(1429897995485)/",
    "LastUpdatedDate": "/Date(1429897995487)/",
    "ParentId": null,
    "ProjectShareOwnerEmail": null,
    "ProjectShareOwnerName": null,
    "SyncClientCreationId": null
}
```

9. If you want to use an external file (this is useful for larger parameters), you must create a file with the extension that API allows (json and xml for this example). For this example we have created 2 files: newProject.xml and newProject.json (the content of files is showed in image below)

```
phantom : x11vnc                                    curl : fish
phantom@ETAQUICHIM-NH03 ~/curl> cat newProject.xml
<ProjectObject>
  <Content>New Project 4</Content>
  <Icon>4</Icon>
</ProjectObject>
phantom@ETAQUICHIM-NH03 ~/curl> cat newProject.json
{
    "Content": "New Project API 2",
    "Icon":4
}
phantom@ETAQUICHIM-NH03 ~/curl>
```

10. Now we will call to the method using these files. Type in terminal:

```
curl -u "username:password" -d @newProject.json
https://todo.ly/api/projects.json | python -mjson.tool
```

the only change was the -d option to reference to an external file using @ symbol (@externalfile)

```
phantom@ETAQUICHIM-NH03 ~/curl> curl -u "taquimon@gmail.com:          " -d @newProject.json https://todo.ly/api/projects.json | python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   461  100   406  100    55    228     30  0:00:01  0:00:01 --:--:--   228
{
    "Children": [],
    "Collapsed": false,
    "Content": "New Project API 2",
    "Deleted": false,
    "Icon": 4,
    "Id": 3428976,
    "IsOwnProject": true,
    "IsProjectShared": false,
    "IsShareApproved": false,
    "ItemOrder": 18,
    "ItemType": 2,
    "ItemsCount": 0,
    "LastSyncedDateTime": "/Date(1429900752405)/",
    "LastUpdatedDate": "/Date(1429900752407)/",
    "ParentId": null,
    "ProjectShareOwnerEmail": null,
    "ProjectShareOwnerName": null,
    "SyncClientCreationId": null
}
phantom@ETAQUICHIM-NH03 ~/curl>
```

11. The image below shows the same example using xml format

```
curl -u "username:password" -d @newProject.xml
https://todo.ly/api/projects.xml
```

```
phantom@ETAQUICHIM-NH03 ~/curl> curl -u "taquimon@gmail.com:          " -d @newProject.xml https://todo.ly/api/projects.xml
<ProjectObject>
  <Id>3428977</Id>
  <Content>New Project 4</Content>
  <ItemsCount>0</ItemsCount>
  <Icon>4</Icon>
  <ItemType>2</ItemType>
  <ParentId p2:nil="true" xmlns:p2="http://www.w3.org/2001/XMLSchema-instance" />
  <Collapsed>false</Collapsed>
  <ItemOrder>19</ItemOrder>
  <Children />
  <IsProjectShared>false</IsProjectShared>
  <IsShareApproved>false</IsShareApproved>
  <IsOwnProject>true</IsOwnProject>
  <LastSyncedDateTime>2015-04-24T18:43:16.1110283Z</LastSyncedDateTime>
  <LastUpdatedDate>2015-04-24T18:43:16.11Z</LastUpdatedDate>
  <Deleted>false</Deleted>
</ProjectObject>
phantom@ETAQUICHIM-NH03 ~/curl>
```

2. AUTOMATED TESTING

a. SOAPUI

i. ABOUT THE TOOL

SoapUI is an open-source web service testing application for service-oriented architectures (SOA) and representational state transfers (REST).

Core features include:

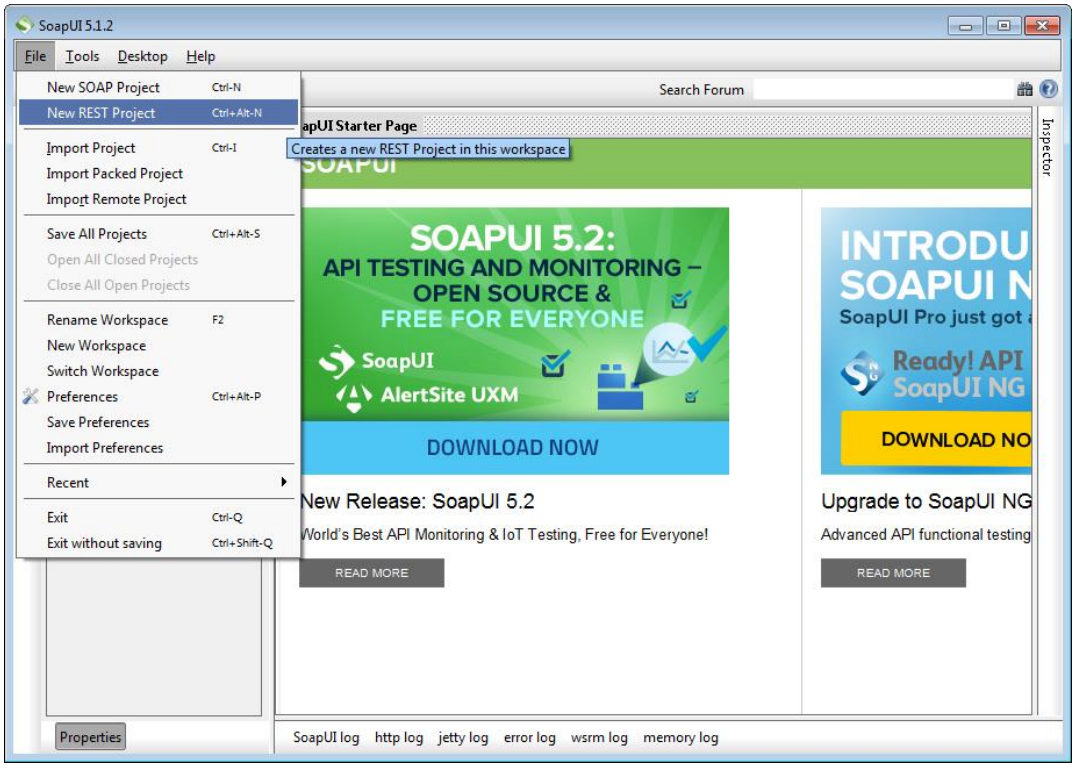
- Web services inspection
- Web services invoking
- Web services development
- Web services simulation and mocking

- Web services functional, load, compliance and security testing

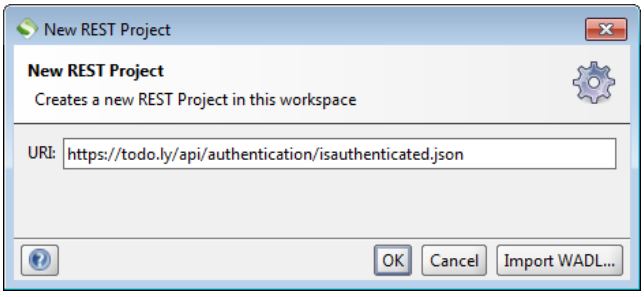
ii. EXAMPLE

We will use todo.ly public API (https://todo.ly/api/project.json). In order to use this API you should create an account on the site.

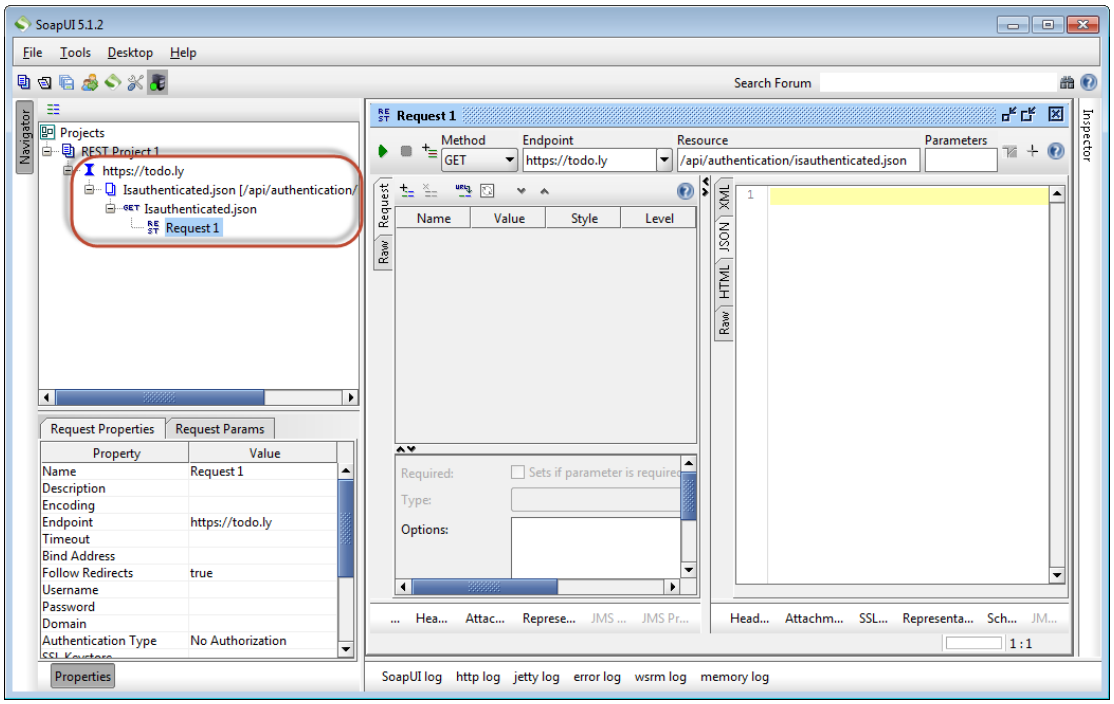
1. Create a new project in SoapUI from file menu:



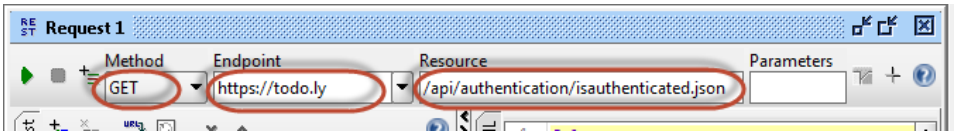
2. Add the resource URL, in this case, we will use the resource to check if we are authenticated (public):



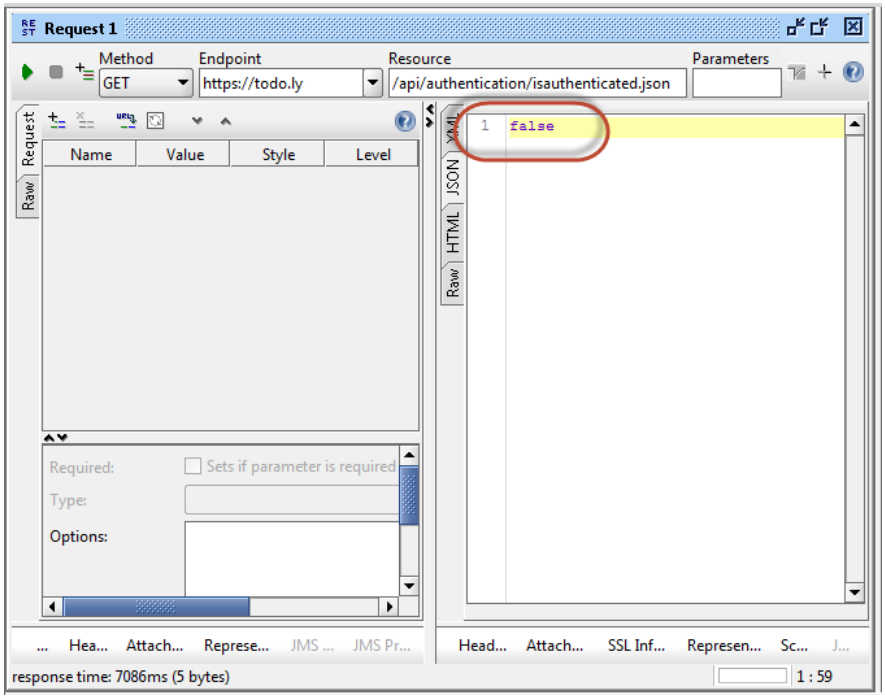
3. You will see something like this (a default test request):



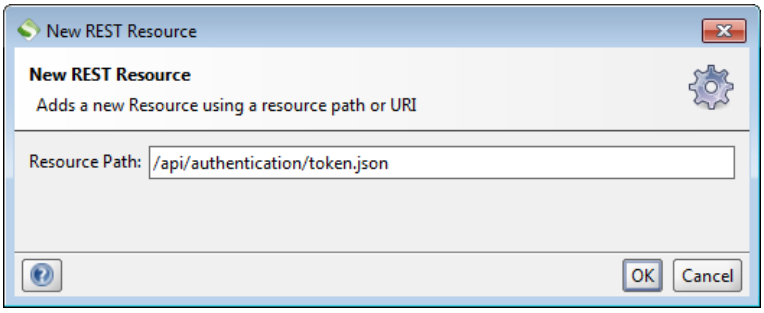
4. Run the test request using the following configuration (according to the API resource, GET in this case):



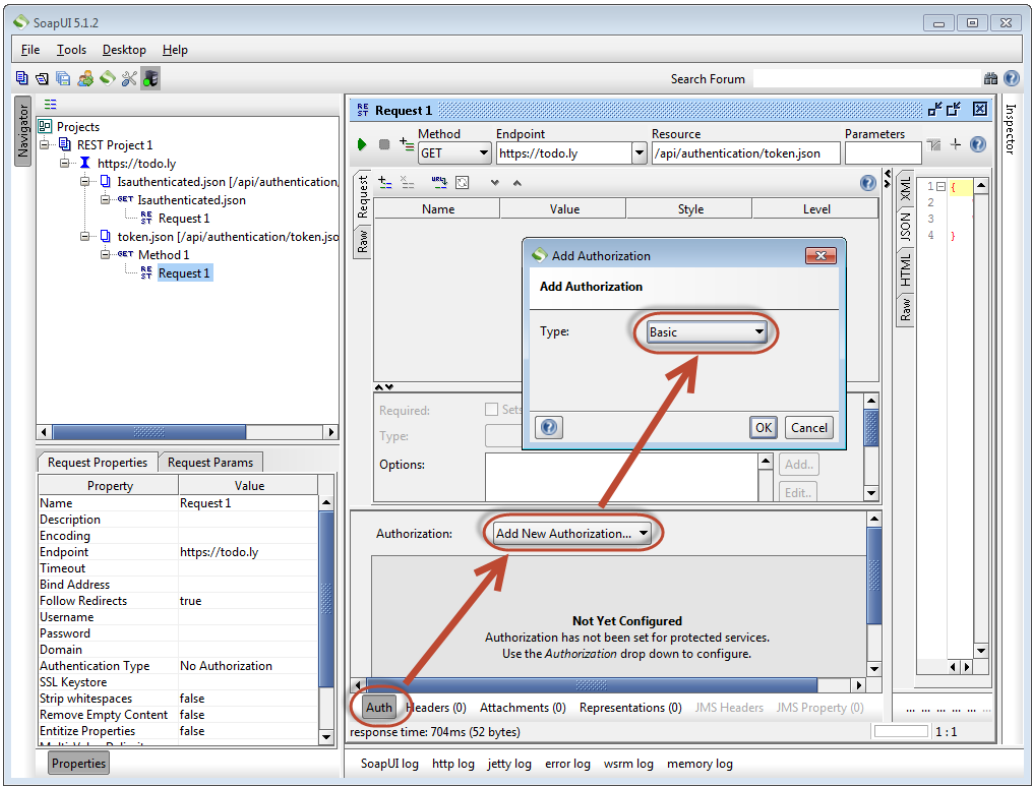
5. You should get a response from the API resource:



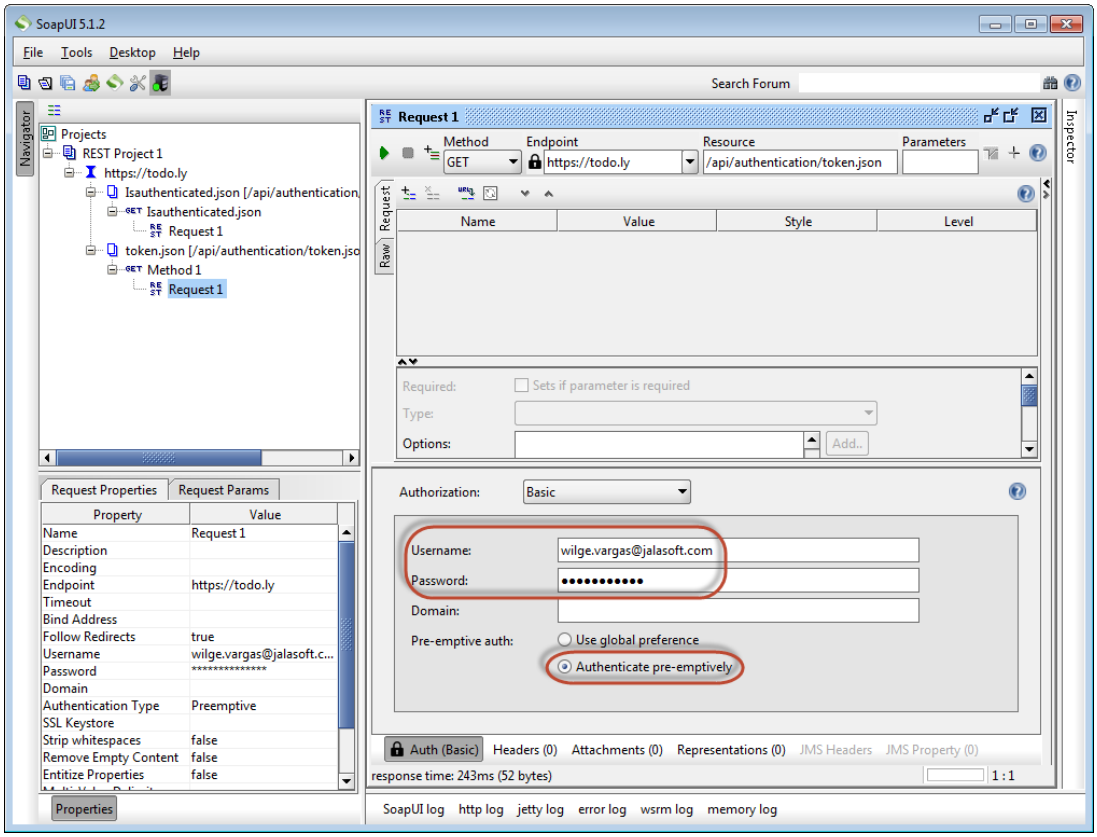
6. Now, let's call to a new resource that needs authentication. First, we need to add the resource:



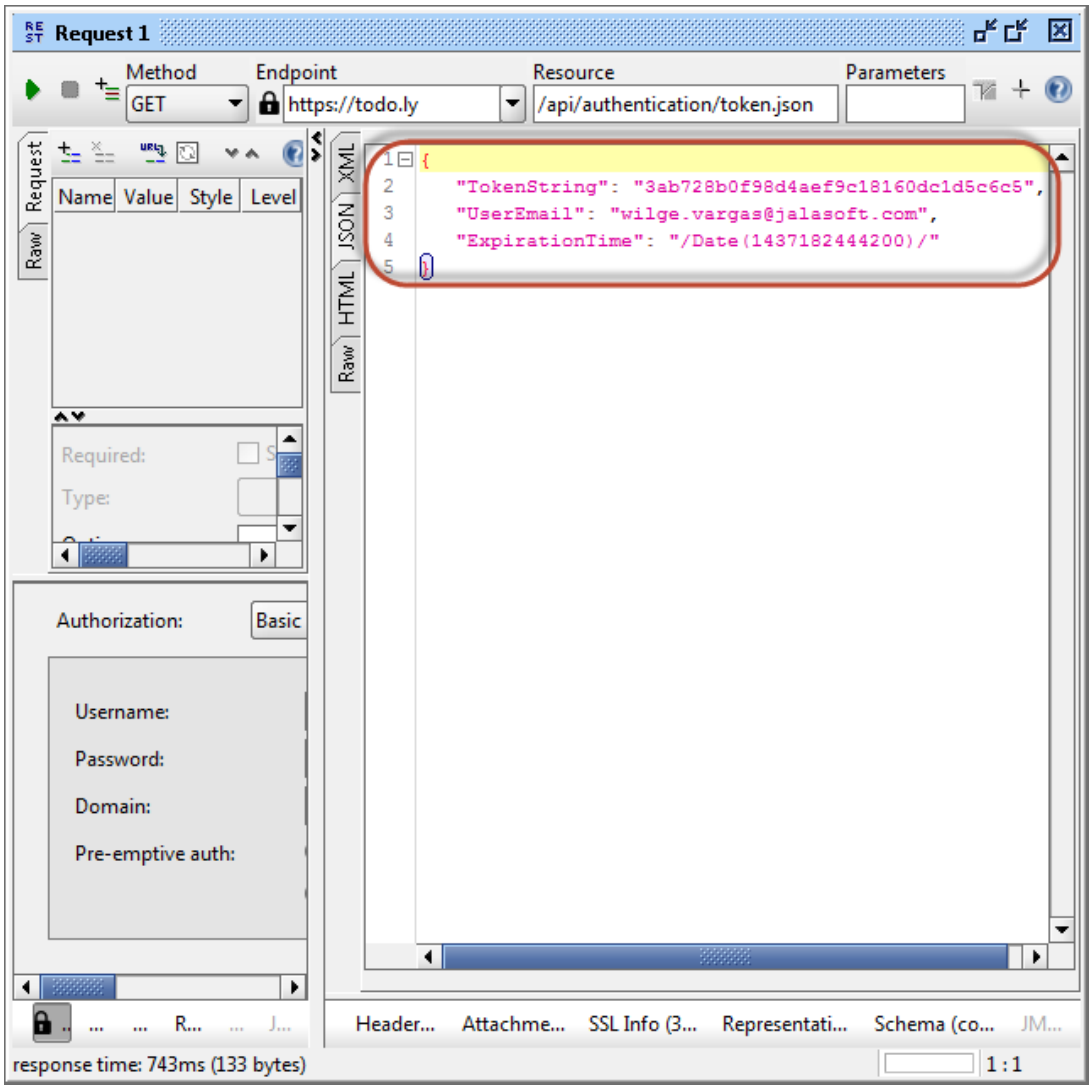
7. We will need to add and Authentication Type, in this case: Basic:



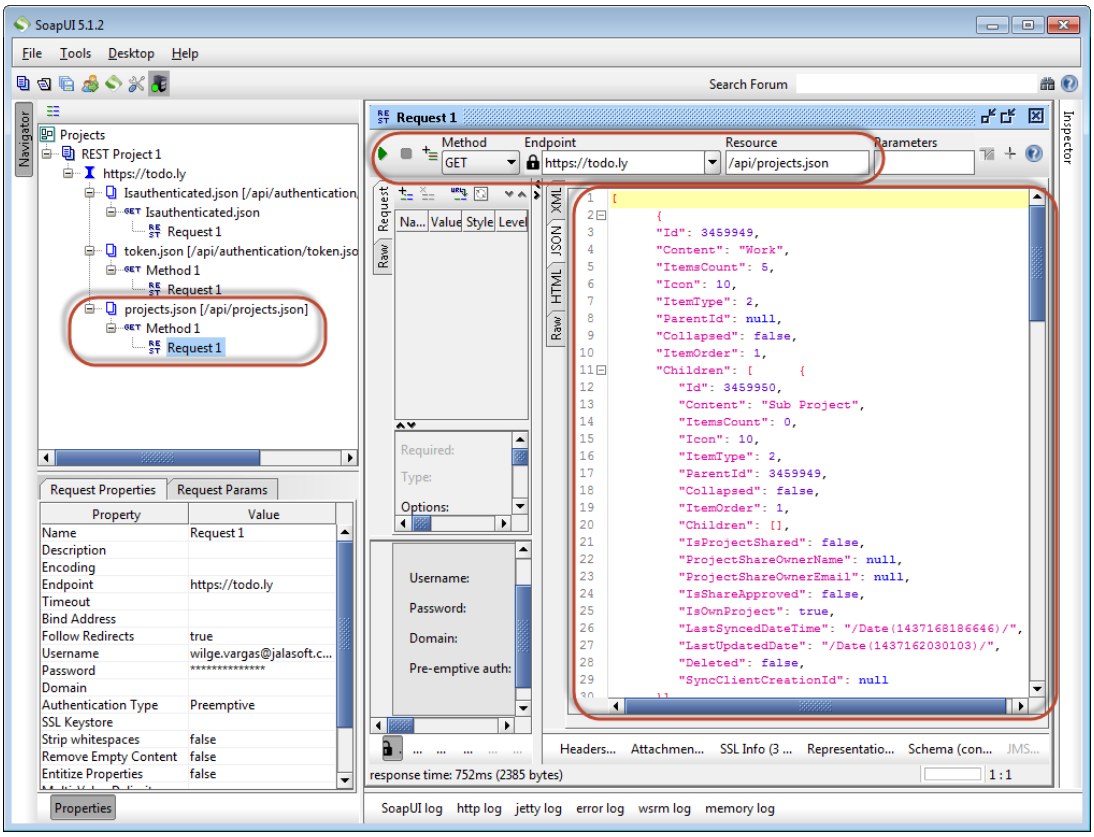
8. Add your credentials:



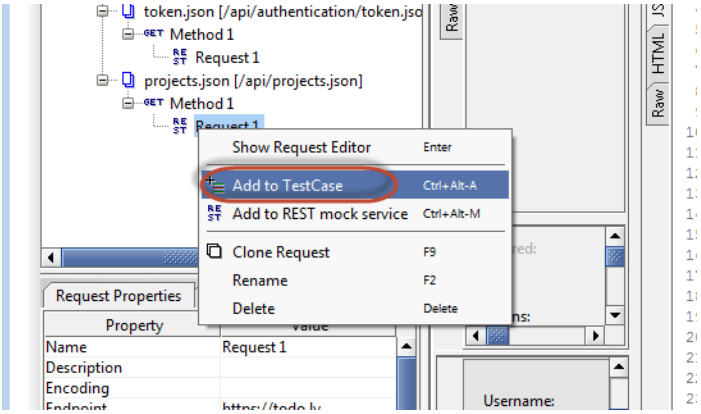
9. Once the Authentication is set up, we can make the call to the API.



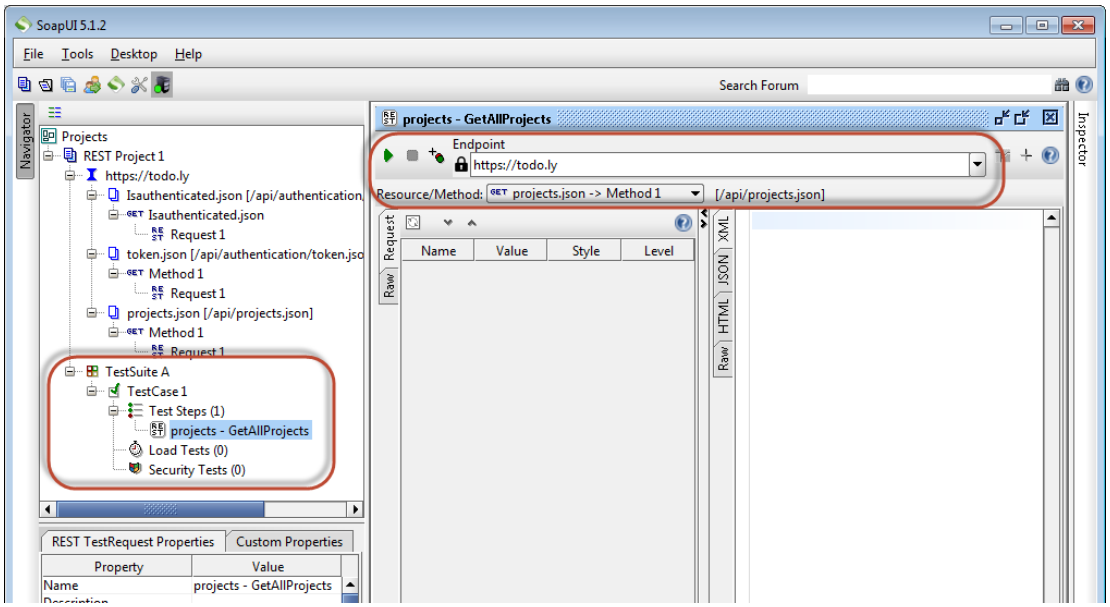
10. Then you can add new resources to the mapped service in SoapUI to run other calls:



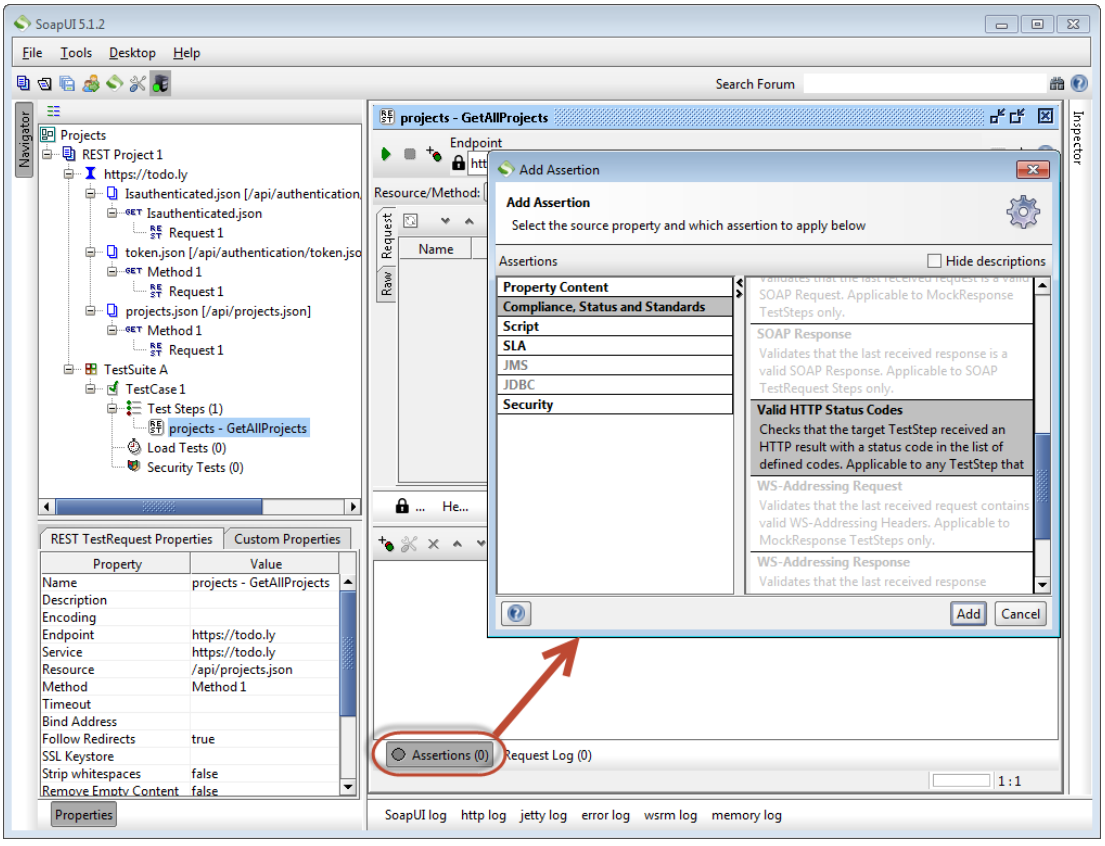
11. For automation purposes, you can create TestCases for each of these resources you added:



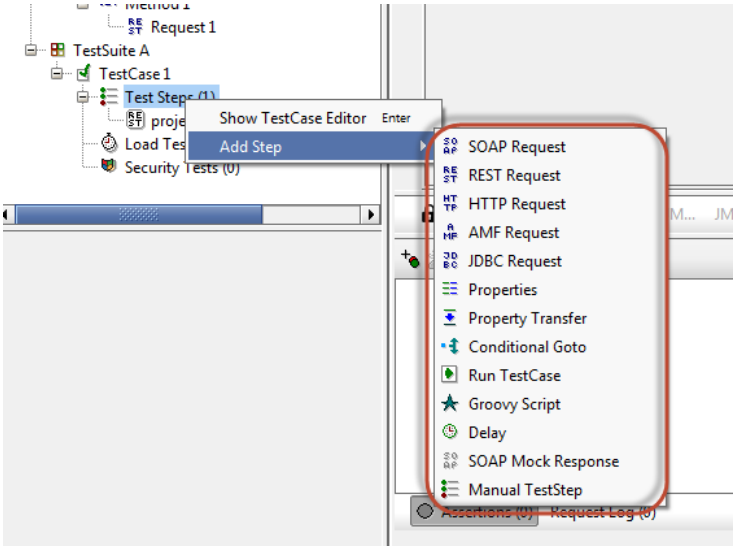
12. It will create a Test Suite and a Test Case and copy the request there. Make sure you put self-explanatory names to the objects.



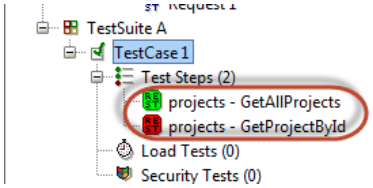
13. You can add basic assertions to each the request, for example: Valid HTTP Status code, response contains something, XPath match, etc.:



14. You can also add different kind of steps to the TestCase, they can be other requests, add more specific validations, loops, DB checks, etc.



15. Once you run the TestCase, you will see all the steps marked as Green=Passed and Red=Failed



16. To call a TestCase from an automated script, you can use SoapUI's testrunner.bat. Check more information here: http://www.soapui.org/test-automation/running-functional-tests.html

b. Frisby and Jasmine-Node (JS)

i. About The Tool

Frisby is a REST API testing framework built on node.js and Jasmine.gem that makes testing API endpoints easy, fast, and fun.

ii. Installation of Frisby

Globally -> *npm install frisby -g*
Locally -> *npm install frisby*

iii. Installation of Jasmine-Node

Globally -> *npm install jasmine-node –g*

iv. Tests Notation

The tests has to have the "spec" in the end of the file name

Example:

*testexample_**spec**.js*

Without this Jasmine will not recognize the file and no test will be run.

v. EXECUTION

The test cases are executed by jasmine-node with the following command line:

Installed globally-> Example: jasmine-node spec\testexample_spec.js

Verbose mode

Example: *jasmine-node spec\testexample_spec.js --verbose*

Report

Example: *jasmine-node spec\testexample_spec.js --junitreport*

Options

There are other options, in order to see them just the command line below

Example: *jasmine-node*

vi. EXAMPLES

The examples written are for the case when jasmine and frisbyjs are installed globally.

The code can be written in any text editor.

The website used for this is Todo.ly, which contains a public REST API that supports XML and JSON.

1. Create a Project (Basic Authentication)

```
// json that specifies the data for the project to be created, the mandatory field it is the name
var project = {
    Content:'Project_examples_frisby',
    Icon: 1
};


frisby.create('Create a simple project')
    .post('https://todo.ly/api/projects.json',
        project,{
            //specifing that the request is sending the data in JSON format
            json:true
        })
    //adding a header called Authorization
    .addHeader('Authorization',"Basic 29yZGluZXMwMDdAZ21haWwuY29tOmNvbnRyb2wxMjM=")
    //validating the response code
    .expectStatus(200)
    //validating the header - content type also can be done for other headers
    .expectHeader('Content-Type','application/json; charset=utf-8')
    //Validating the JSONTypes, here are some examples
    .expectJSONTypes({
        Id:Number,
        Content:String,
        Icon:Number,
        Children:Array,
        Deleted:Boolean
    })
    //validating the JSON data
    .expectJSON({
        Content:project.Content,
        Icon:project.Icon,
        Deleted:false
    })
    //this is executed once the response is received
    .after(function(err, res,body){
        //Here is printing each one of the parameters
        console.log('##########################################' );
        console.log('Create error:' + err);
        console.log('------------------------------------------' );
        console.log('Create response: ' + res.headers['content-type']);
        console.log('Create response: ' + res.headers['content-length']);
        console.log('------------------------------------------' );
        console.log('Create project Id: ' + body.Id);
        console.log('Create project Content: ' + body.Content);
        console.log('##########################################' );
    })
//this launches the test
.toss();
```

2. Get a Project (Basic Authentication)

```
var projectCreated = {
    Id : 3434775,
    Content:'Project_examples_frisby'
};

/*
 *  Get a project, in order to get the information from a project it is needed to know the Id of the project
 *  In this case let's use the same project used for the update
*/

frisby.create('Get a project')
    .get('https://todo.ly/api/projects/'+projectCreated.Id+'.json')
    .expectJSON({
        Id:projectCreated.Id,
        Content:projectCreated.Content,
        Deleted:false
    })
    .afterJSON(function(response){
        console.log('########################################' );
        console.log('Get project Id: ' + response.Id);
        console.log('Get project Content: ' + response.Content);
        console.log('########################################' );
    })
    //this prints all the json
    .inspectJSON()

.toss();
```

3. Update a Project (Basic Authentication)

```
/*
    Update a project, in order to edit a project it is needed to know the Id of the project
    to be edited
*/
var projectCreated = {
    Id : 3434775,
    Content:'Project_examples_frisby_updated'
};

frisby.create('Update a project')
    .put('https://todo.ly/api/projects/'+projectCreated.Id+'.json',{
        //just updating the name of the project, this also allows to change other attributes
        Content:projectCreated.Content
    },{
        json:true
    })
    .addHeader('Authorization','Basic Z29yZG1uZXMwMDdAZ21haWwuY29tOmNvbnRyb2wxMjM')
    .expectJSON({
        Id:projectCreated.Id,
        Content:projectCreated.Content,
        Deleted:false
    })
    //this is executed once the response is received
    .afterJSON(function(response){
        console.log('############################################' );
        console.log('Update project Id: ' + response.Id);
        console.log('Update project Content: ' + response.Content);
        console.log('############################################' );
    })
.toss();
```

4. Delete a Project (Basic Authentication)

```
var projectCreated = {
    Id : 3434775,
    Content:'Project_examples_frisby_updated'
};


/**
 *  Delete a project, in order to delete a project it is needed to know the Id of the project
 *  In this case let's use the same project used for the update and get examples
*/
frisby.create('Delete a project')
    .delete('https://todo.ly/api/projects/'+projectCreated.Id+'.json')
    .addHeader('Authorization',"Basic Z29yZGluZXMwMDdAZ21haWwuY29tOmNvbnRyb2wxMjM=")
    .expectJSON({
        Id:projectCreated.Id,
        Content:projectCreated.Content,
        Deleted:true
    })
    .inspectJSON()
    .after(function(err,response,body){
        console.log('##########################################' );
        console.log('Deleted error:' + err);
        console.log('------------------------------------------' );
        console.log('Deleted response: ' + res.headers['content-type']);
        console.log('Deleted response: ' + res.headers['content-length']);
        console.log('------------------------------------------' );
        console.log('Delete project Id: ' + body.Id);
        console.log('Delete project Content: ' + body.Content);
        console.log('Delete project Deleted: ' + body.Deleted);
        console.log('##########################################' );
    })
.toss();
```

5. **Use of Tokens instead of Basic Authentication**

   The only change is in the header added for the authentication, instead of ***authorization*** use ***token,*** with the corresponding token returned by the site (review the section about the Authentication API Methods).

```
//adding a header called token
.addHeader('token',"d4efdf30c002439689fb125c54a57094")
```

vii. References

- http://frisbyjs.com/docs/api/

c. RestAssured

i. About the Tool

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. Not only is it easy to use and get started with but it's also built to scale to more advanced use cases using detailed configuration, filters, specifications. IT IS SIMPLE mainly thanks to A syntax, given-when-then

ii. Configuration

- **Maven / Gradle Users**

  Add the following dependency to your pom.xml: ***REST Assured***
  Includes JsonPath and XmlPath

Maven:

```
<dependency>
    <groupId>com.jayway.restassured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>2.4.1</version>
    <scope>test</scope>
</dependency>
```

Gradle:

```
testCompile 'com.jayway.restassured:rest-assured:2.4.1'
```

- **Non Maven / Gradle Users**

  Download REST Assured.

  Extract the distribution zip file and put the jar files in your class-path.

iii. EXAMPLES

The website used for this is Todo.ly, which contains a public REST API that supports XML and JSON.

1. Create a Project (Basic Authentication)

```
String projectName = "Chelsea";
int icon = 1;
boolean deleted = false;

JSONObject jsonObject = new JSONObject();
jsonObject.put("Content", projectName);
jsonObject.put("Icon", icon);

RestAssured.given().
    headers("Authorization",
        "Basic Y2FybGVkcmlzc0BnbWFpbC5jb206UEBzc3cwcmQ=").
    contentType(ContentType.JSON).
    body(jsonObject.toJSONString()).
when().
    post("https://todo.ly/api/projects.json").
then().
    assertThat().statusCode(200).
    assertThat().contentType(ContentType.JSON).
    assertThat().body("Content", CoreMatchers.equalTo(projectName)).
    assertThat().body("Icon", CoreMatchers.equalTo(icon)).
    assertThat().body("Deleted", CoreMatchers.equalTo(deleted));
```

2. Get a Project (Basic Authentication)

```
String projectID = "3461950";
String expectedProjectName = "Chelsea";
int expectedIcon = 1;
boolean deleted = false;

RestAssured.given().
    headers("Authorization",
```

```
            "Basic Y2FybGVkcmlzc0BnbWFpbC5jb206UEBzc3cwcmQ=").
    when().
        get("https://todo.ly/api/projects/" + projectID + ".json").
    then().
        assertThat().statusCode(200).
        assertThat().contentType(ContentType.JSON).
        assertThat().body("Content", CoreMatchers.equalTo(expectedProjectName)).
        assertThat().body("Icon", CoreMatchers.equalTo(expectedIcon)).
        assertThat().body("Deleted", CoreMatchers.equalTo(deleted));
```

3. Update a Project (Basic Authentication)

```
String projectID = "3461950";
String newProjectName = "Chelsea Updated";
int newIcon = 3;
boolean deleted = false;

JSONObject jsonObject = new JSONObject();
jsonObject.put("Content", newProjectName);
jsonObject.put("Icon", newIcon);

RestAssured.given().
        headers("Authorization", "Basic Y2FybGVkcmlzc0BnbWFpbC5jb206UEBzc3cwcmQ=").
        contentType(ContentType.JSON).
        body(jsonObject.toJSONString()).
    when().
        put("https://todo.ly/api/projects/" + projectID + ".json").
    then().
        assertThat().statusCode(200).
        assertThat().contentType(ContentType.JSON).
        assertThat().body("Content", CoreMatchers.equalTo(newProjectName)).
        assertThat().body("Icon", CoreMatchers.equalTo(newIcon)).
        assertThat().body("Deleted", CoreMatchers.equalTo(deleted));
```

4. Delete a Project (Basic Authentication)

```
String projectID = "3461950";
String projectName = "Chelsea";
int icon = 3;
boolean deleted = true;

RestAssured.given().
        headers("Authorization",
            "Basic Y2FybGVkcmlzc0BnbWFpbC5jb206UEBzc3cwcmQ=").
    when().
        delete("https://todo.ly/api/projects/" + projectID + ".json").
    then().
        assertThat().statusCode(200).
        assertThat().contentType(ContentType.JSON).
        assertThat().body("Content", CoreMatchers.equalTo(projectName)).
        assertThat().body("Icon", CoreMatchers.equalTo(icon)).
        assertThat().body("Deleted", CoreMatchers.equalTo(deleted));
```

iv. REFERENCES

- https://github.com/jayway/rest-assured/wiki/GettingStarted
- https://github.com/jayway/rest-assured/wiki/Usage

d. JMETER

i. ABOUT THE TOOL

JMeter is open source testing software. It is 100% pure Java application for load and performance testing. It may be used to test performance both on static and dynamic resources (Webservices (SOAP/REST) and It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server/script/object behavior under heavy concurrent load.

ii. INSTALLATION

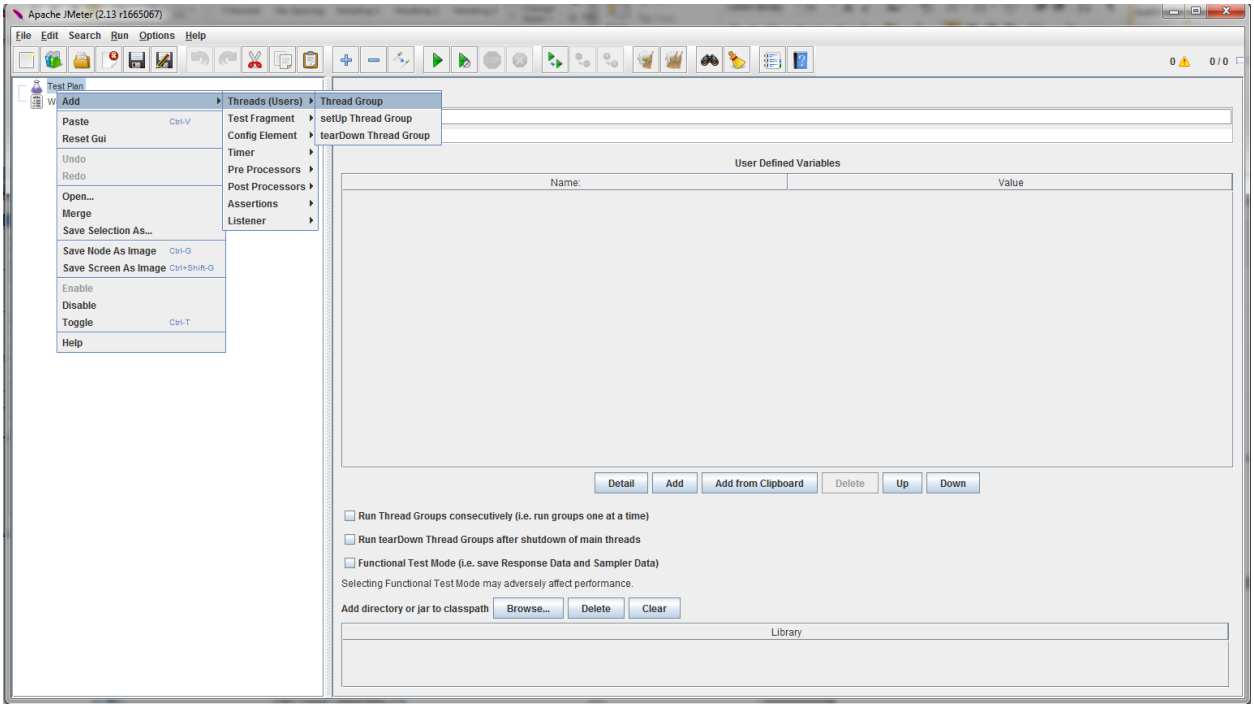
- Needs JDK 1.6 or above installed on the machine
- Download JMeter from http://jmeter.apache.org/download_jmeter.cgi
- After downloading JMeter go to the bin directory and click on the following file:

OS	Output
Windows	jmeter.bat
Linux	jmeter.sh
Mac	jmeter.sh

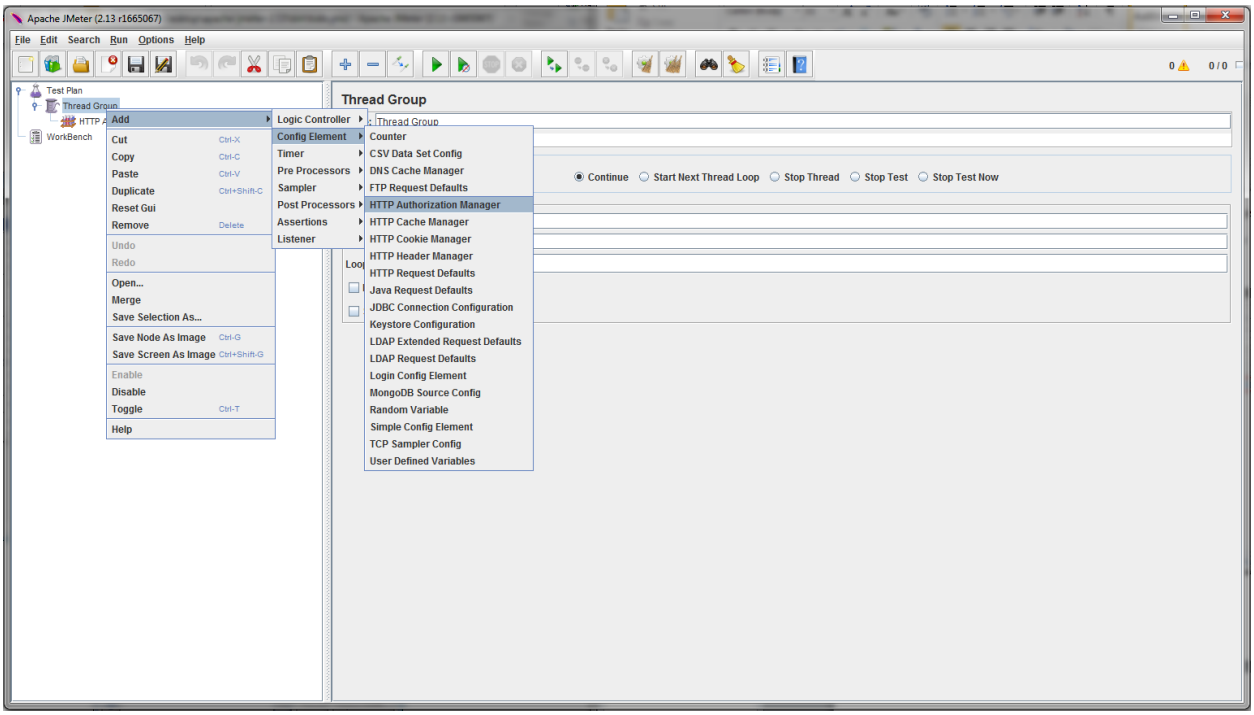
iii. EXAMPLE: API CALLS

We will use todo.ly public API (https://todo.ly/api/projects.json). In order to use this API you should create an account on the site.

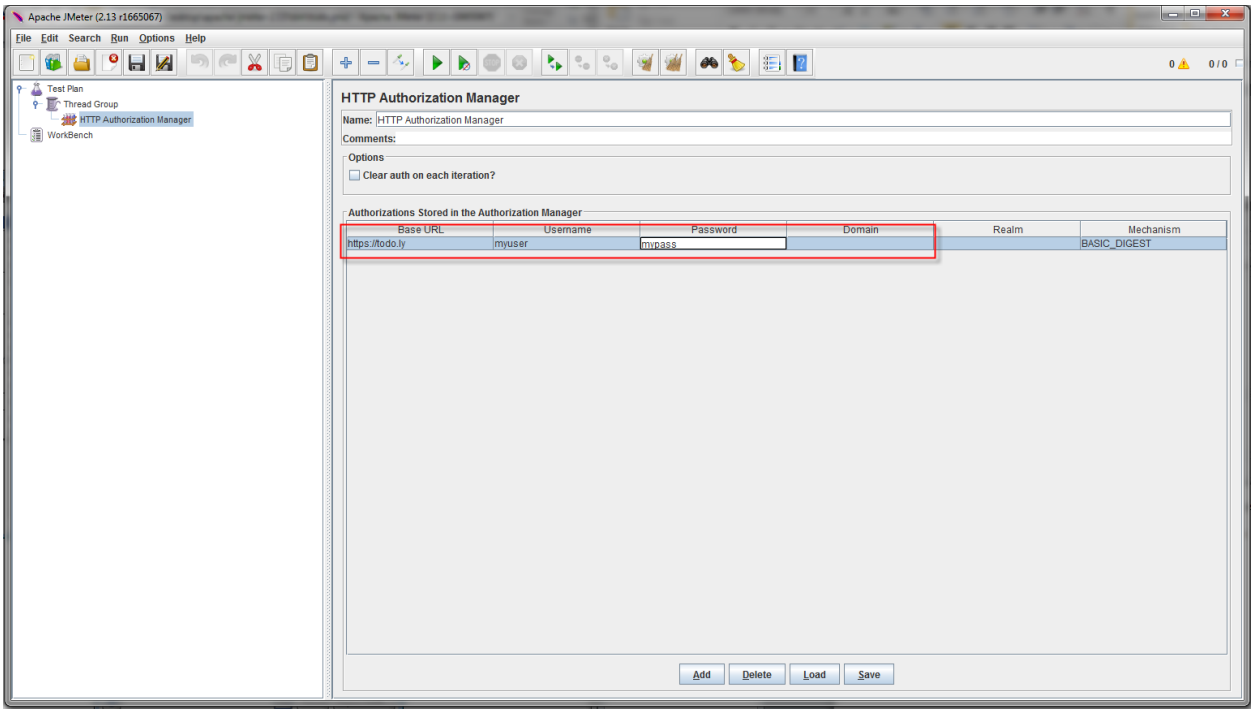
1. Open JMeter and add a new Thread Group (this is our test suit)



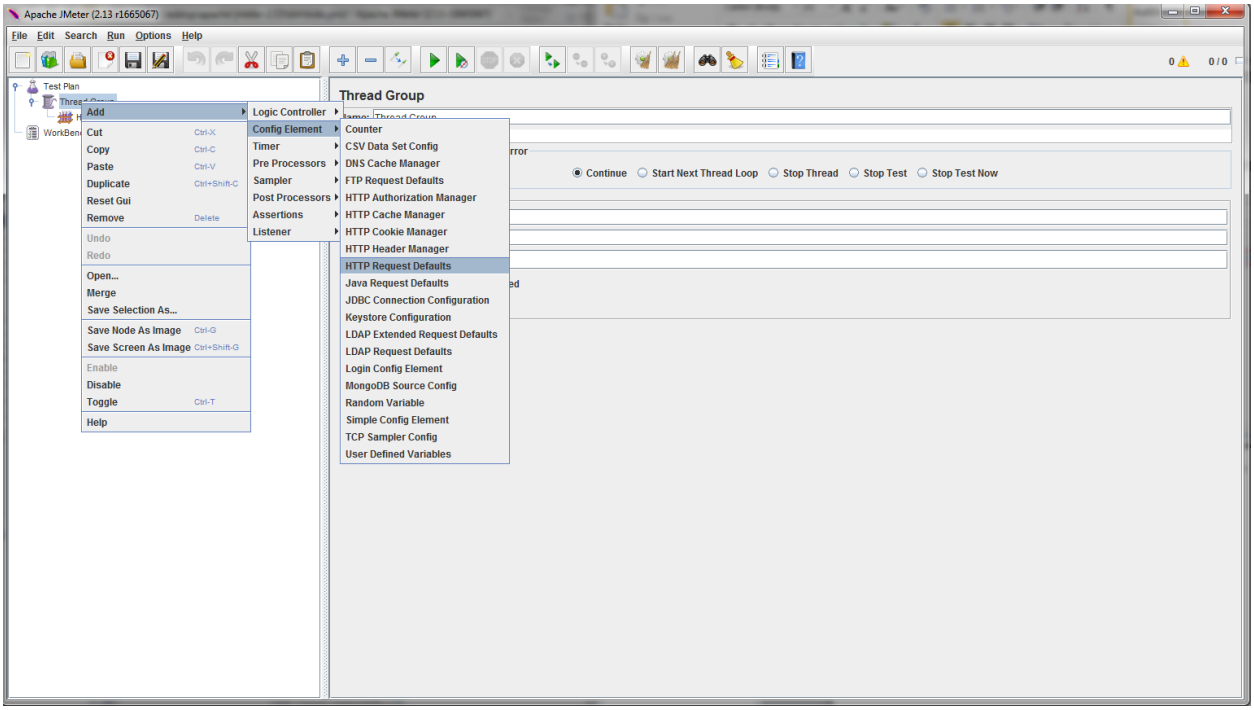
2. Configure the number the users, the ramp-up period and the loops you wish for your test
3. On the created Thread Group add a HTTP Authorization Manager



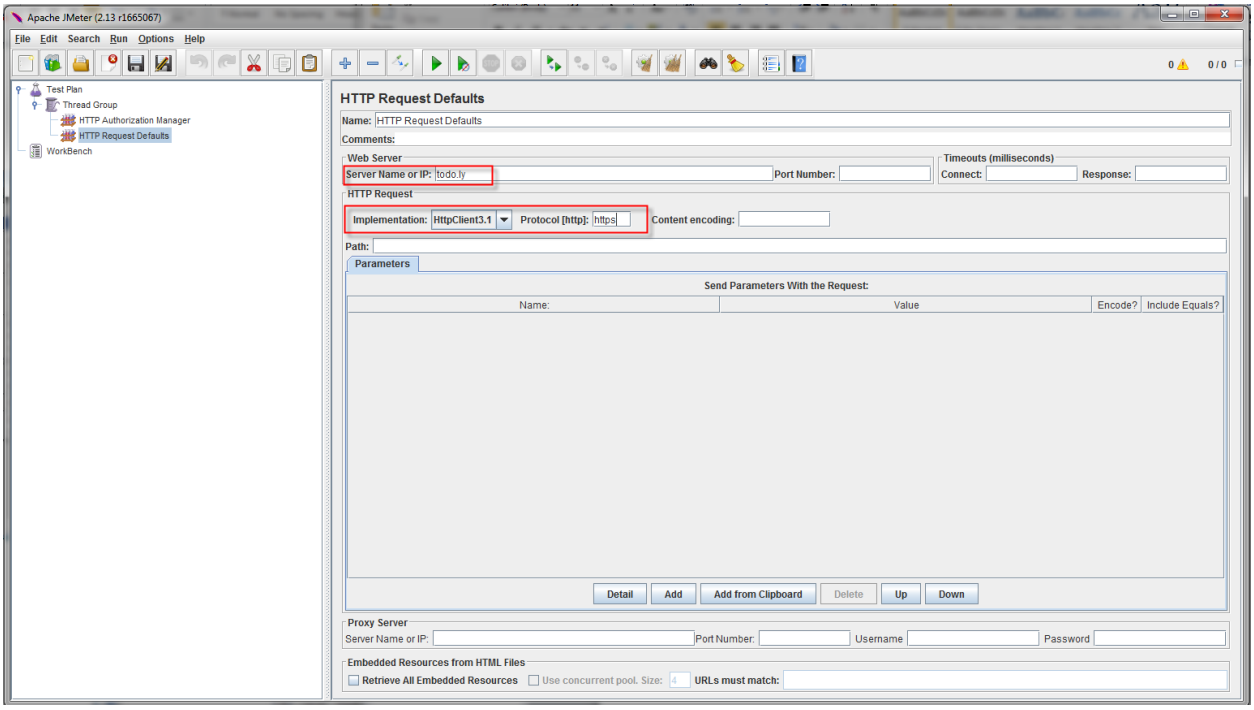
4. Add the URL and your username and password



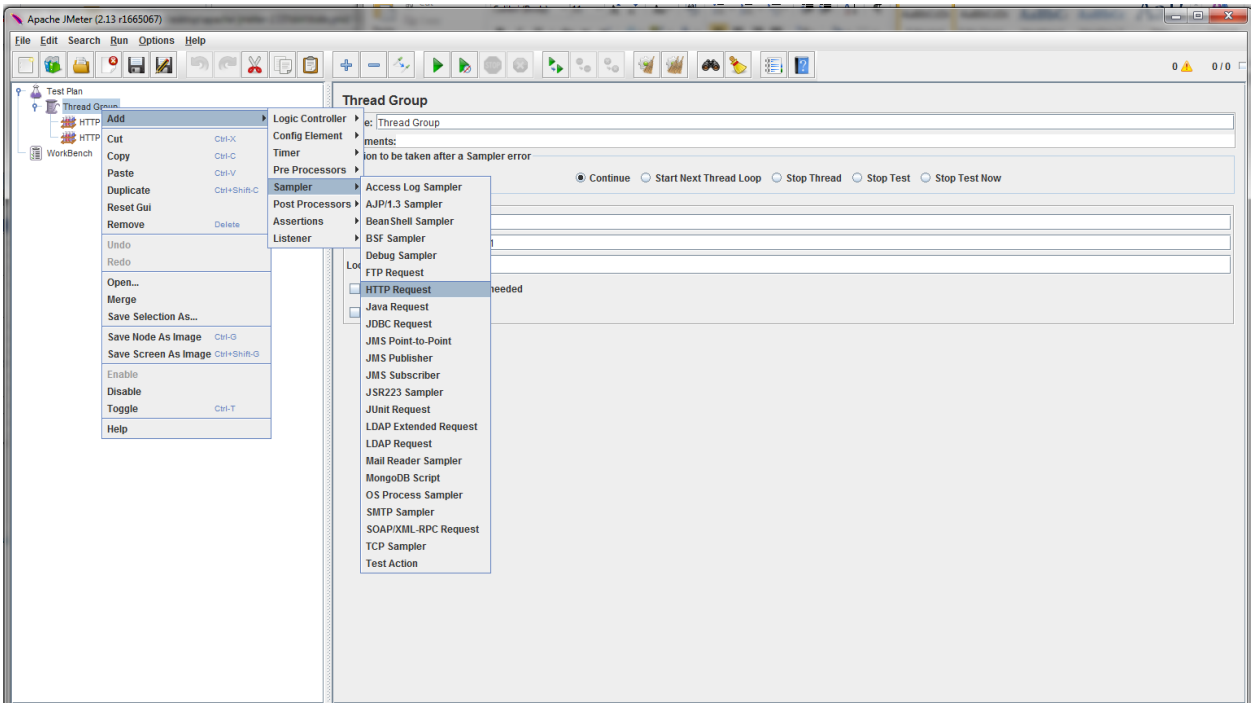
5. On the created Thread Group add a HTTP Request Defaults



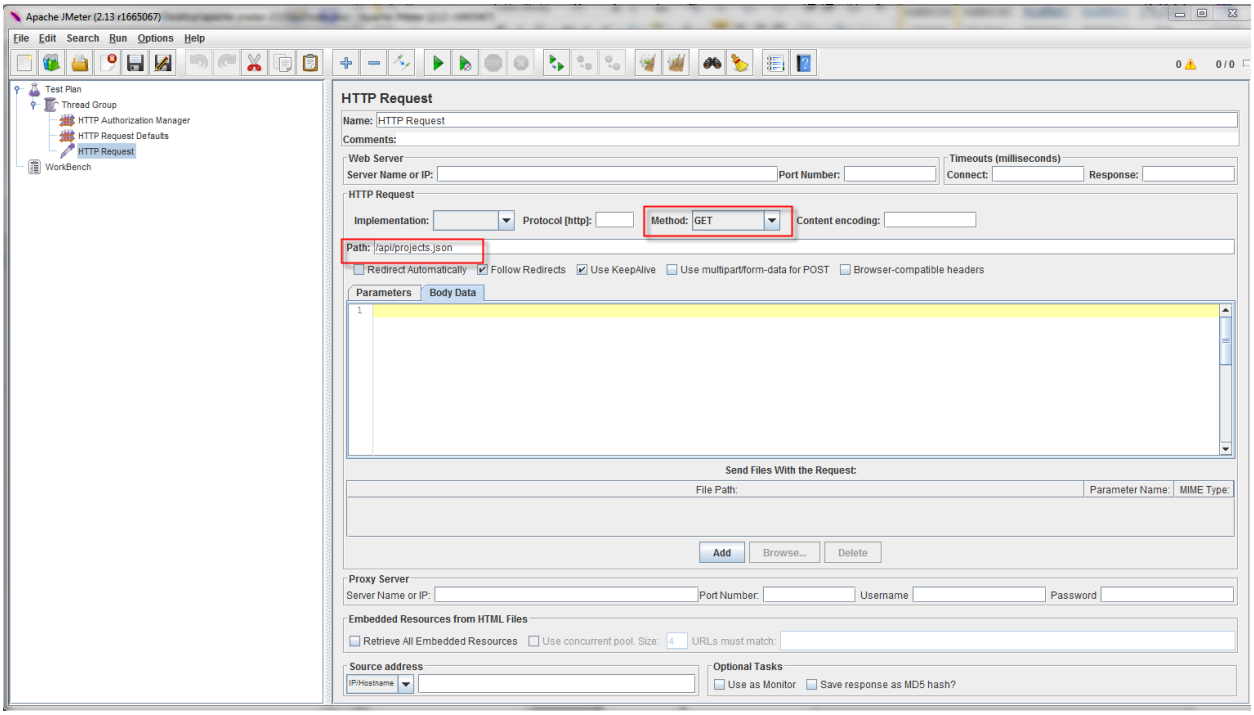
6. Configure the server the protocol and the implementation to use for the request



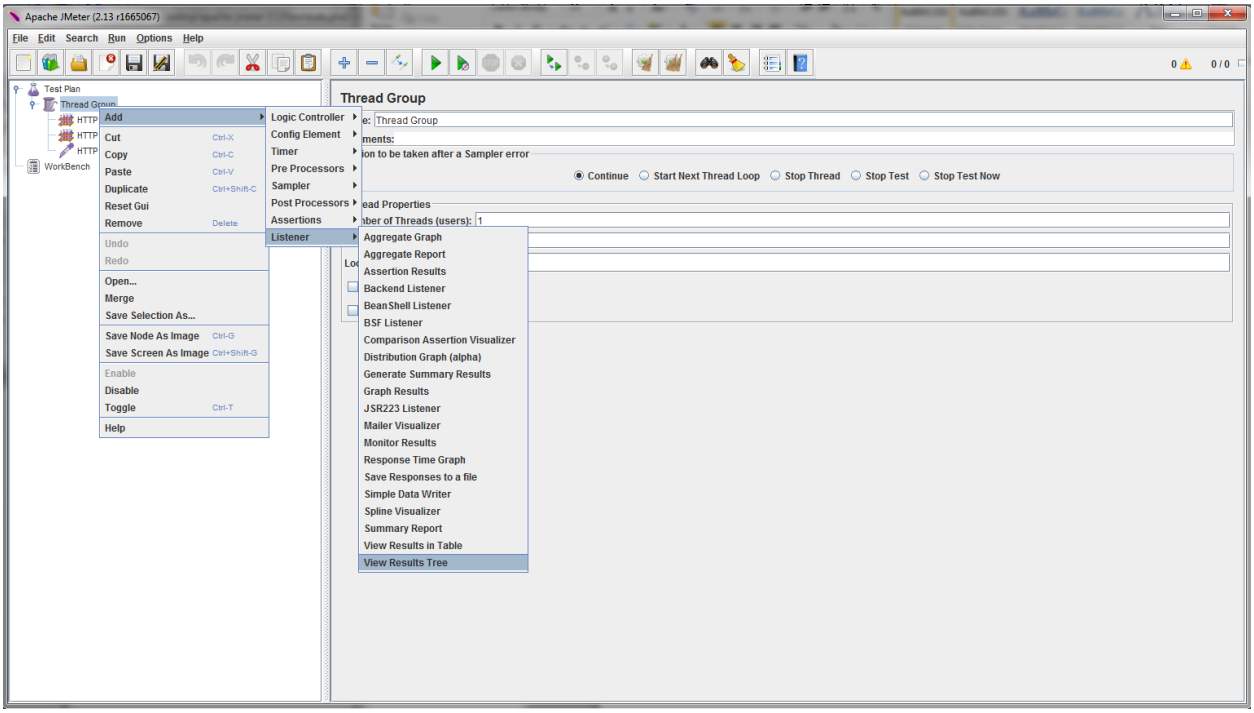
7. On the created Thread Group add a HTTP Request



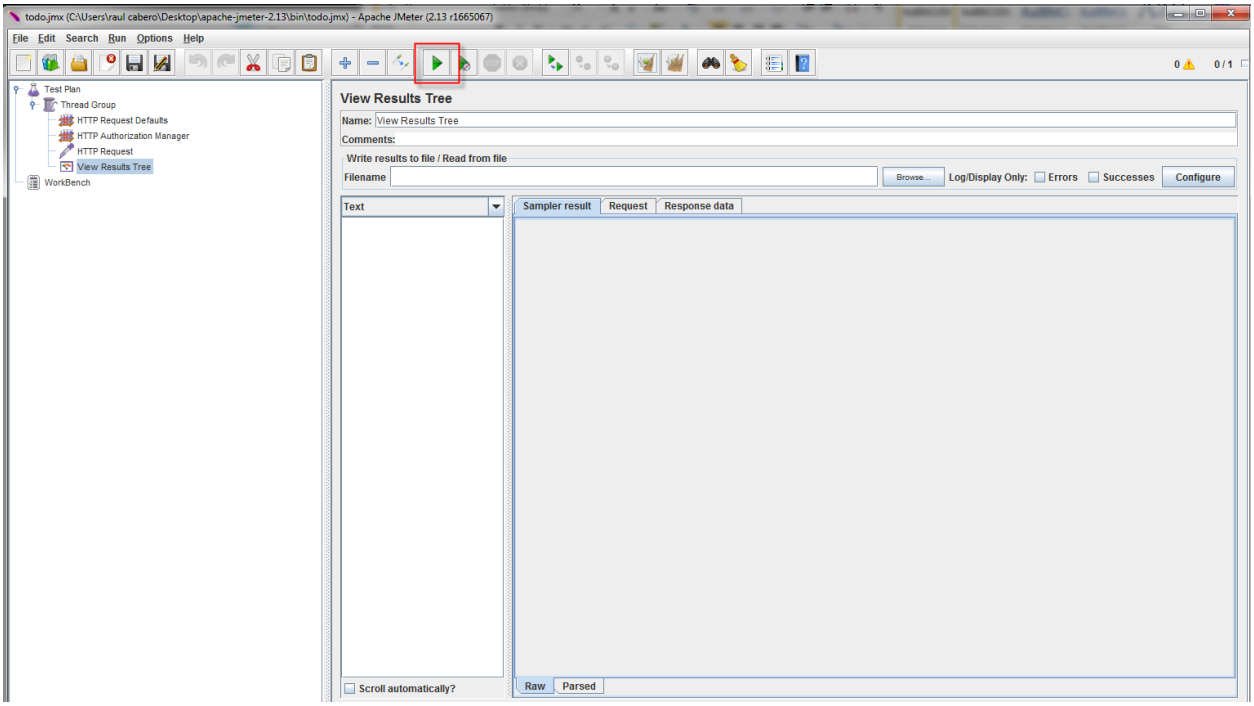
8. Configure the path and method for the request (note: for a Post request you can add the body of the request on Body Data field)



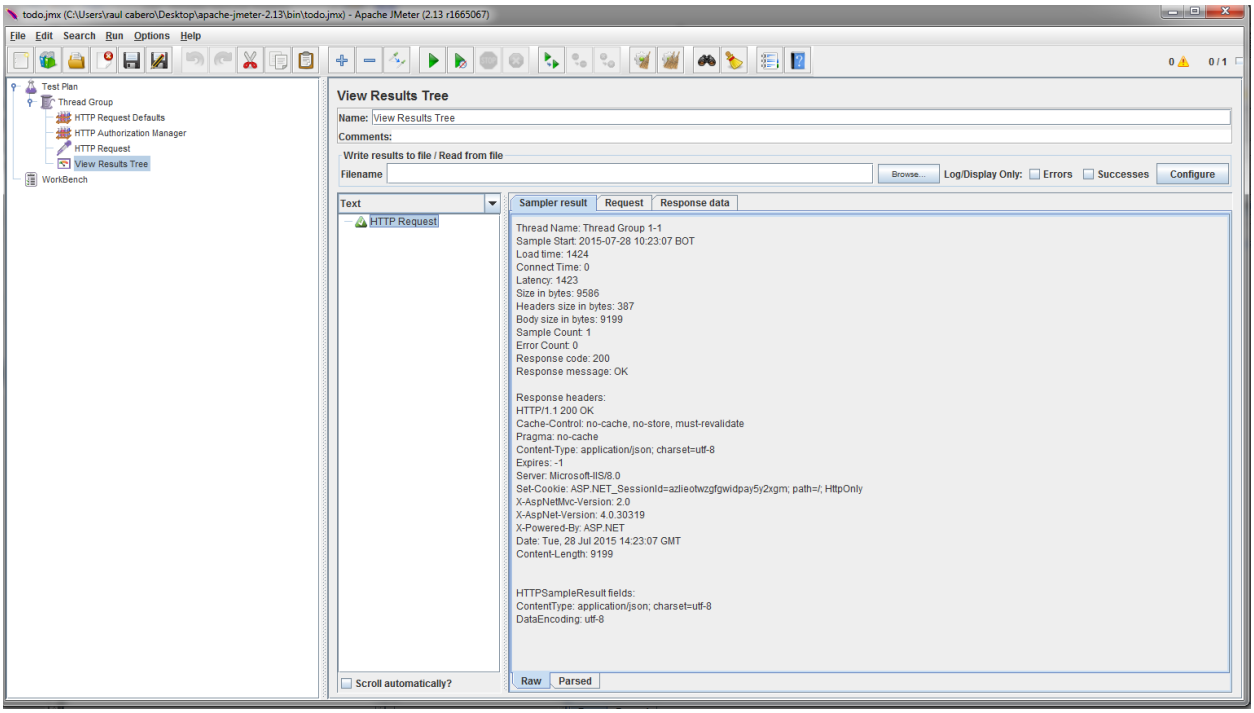
9. On the created Thread Group add a View result Tree in order to see the results of the execution

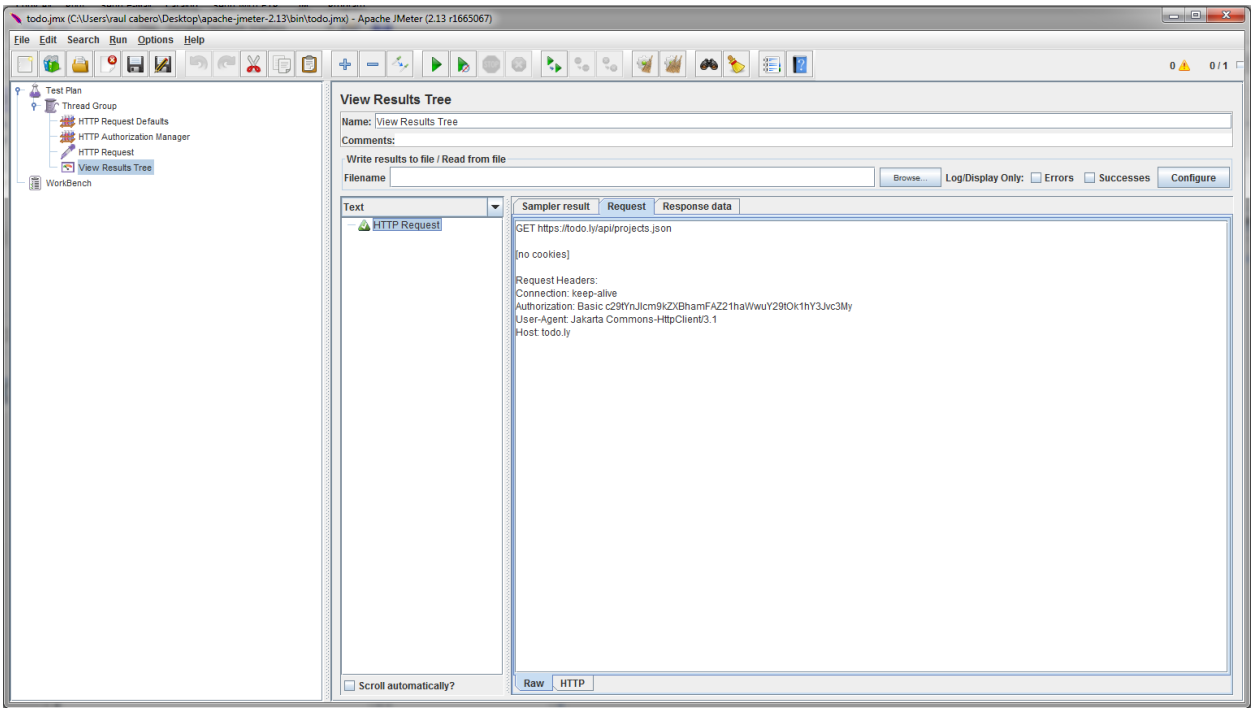


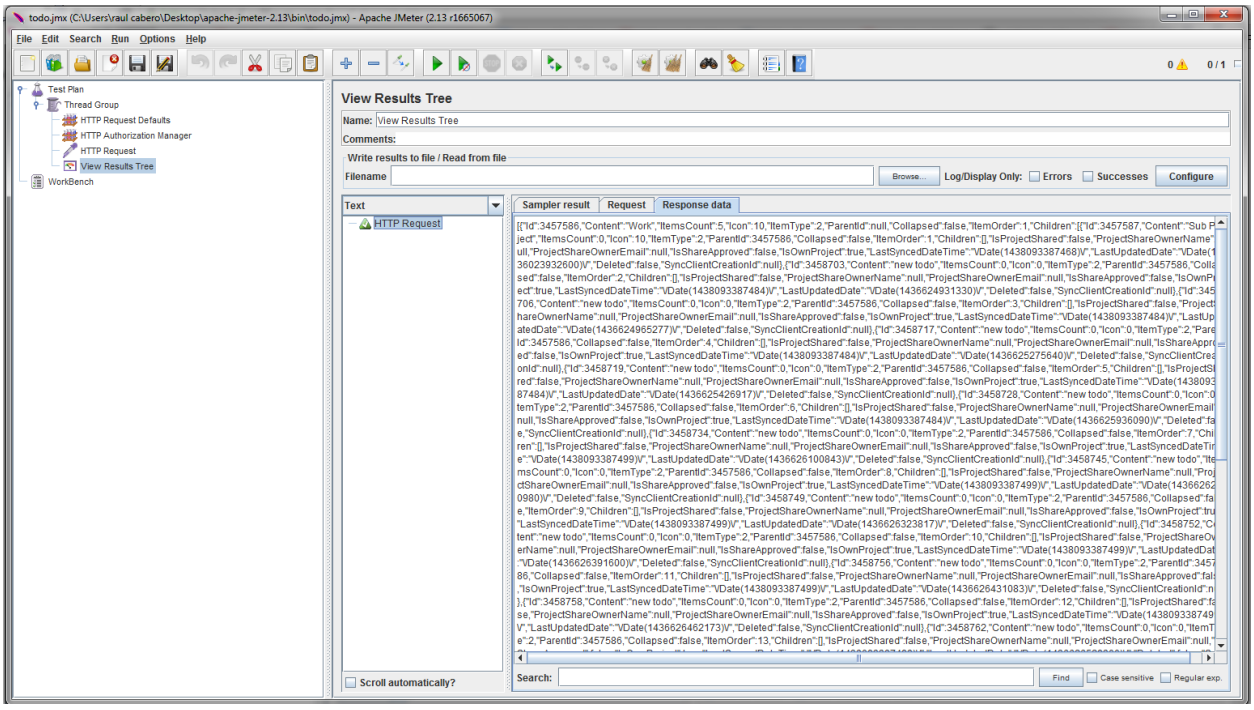
10. Run the test



11. Check the results



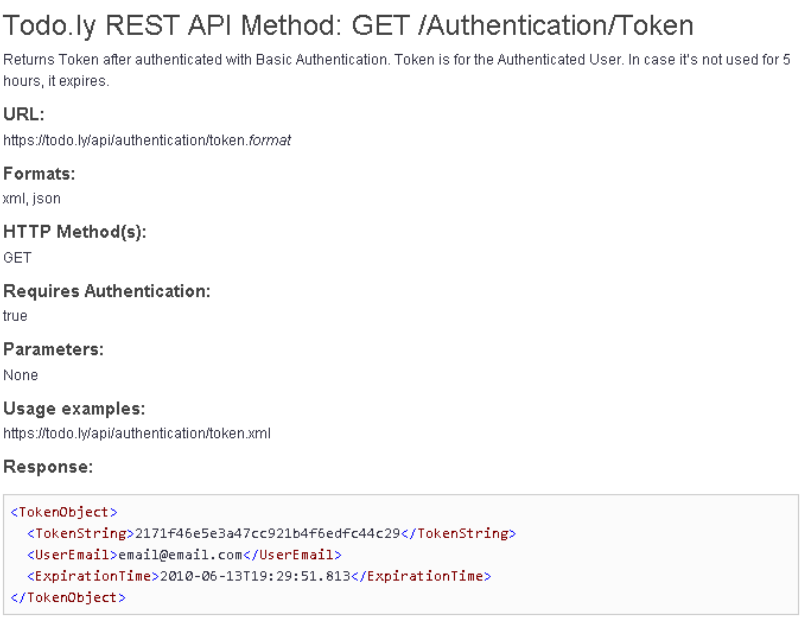




iv. EXAMPLE: PERFORMANCE TEST

We are going to use todo.ly public API (https://todo.ly/api/projects.json) to show this example:

We will use a first end point to get the token,



Todo.ly REST API Method: GET /Authentication/Token

Returns Token after authenticated with Basic Authentication. Token is for the Authenticated User. In case it's not used for 5 hours, it expires.

URL:
https://todo.ly/api/authentication/token.*format*

Formats:
xml, json

HTTP Method(s):
GET

Requires Authentication:
true

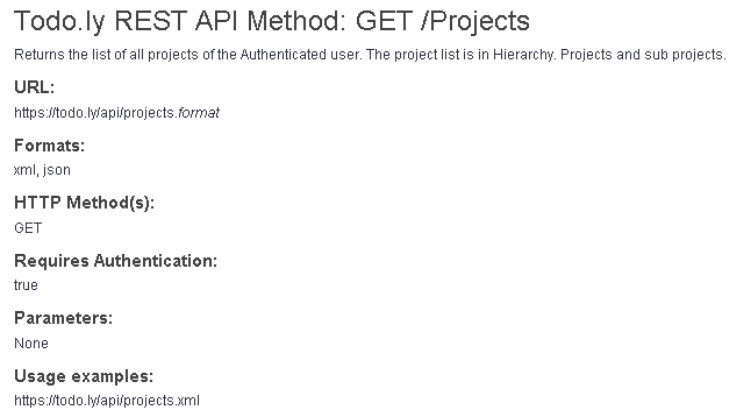
Parameters:
None

Usage examples:
https://todo.ly/api/authentication/token.xml

Response:

```
<TokenObject>
  <TokenString>2171f46e5e3a47cc921b4f6edfc44c29</TokenString>
  <UserEmail>email@email.com</UserEmail>
  <ExpirationTime>2010-06-13T19:29:51.813</ExpirationTime>
</TokenObject>
```

The second end point is to get all projects, but we will use the token obtained in the previous call to authenticate:



Todo.ly REST API Method: GET /Projects

Returns the list of all projects of the Authenticated user. The project list is in Hierarchy. Projects and sub projects.

URL:
https://todo.ly/api/projects.*format*

Formats:
xml, json

HTTP Method(s):
GET

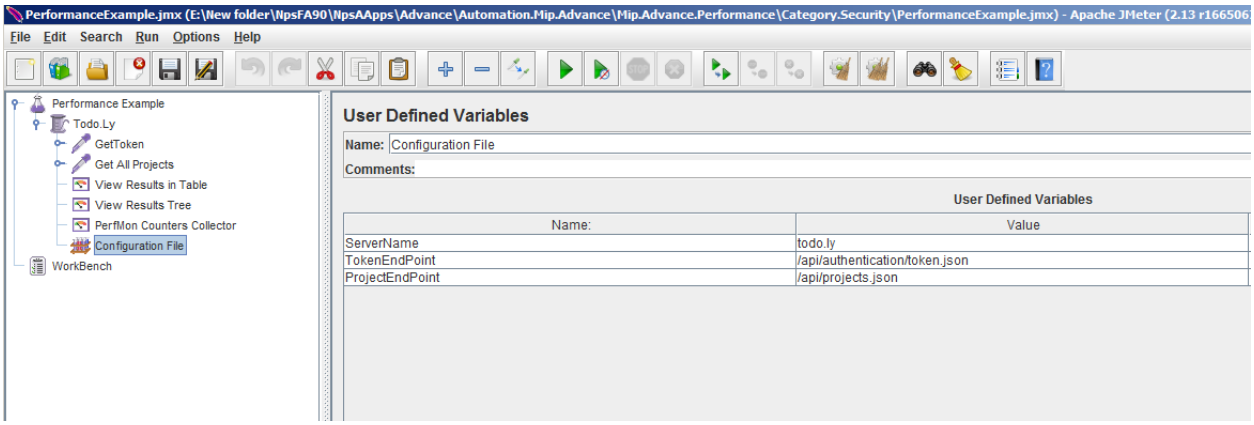
Requires Authentication:
true

Parameters:
None

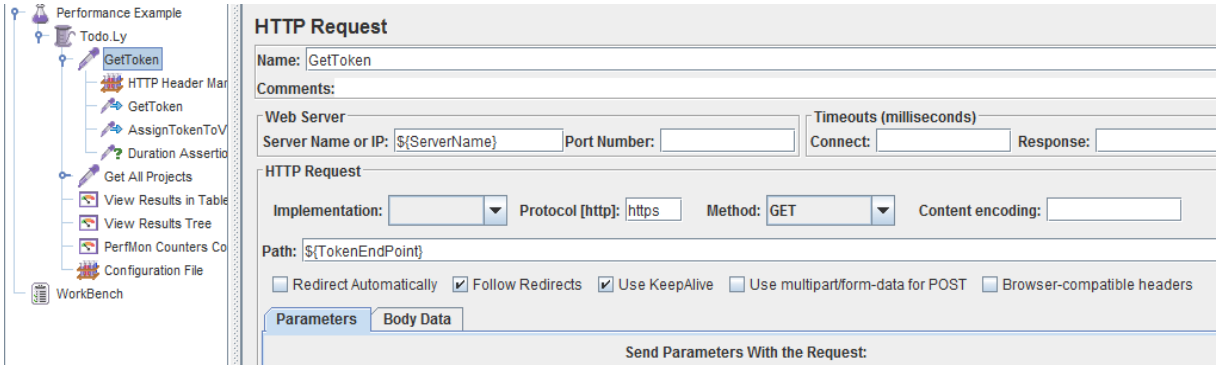
Usage examples:
https://todo.ly/api/projects.xml

Then we can create a Jmeter file (jmx) with 1 Sample, 2 thread Group (each end point and its parameters), Listeners (to see the results). Also we can have all the end point in the configuration file (this dependents on the framework and the way we want to sort the endpoint and inputs).

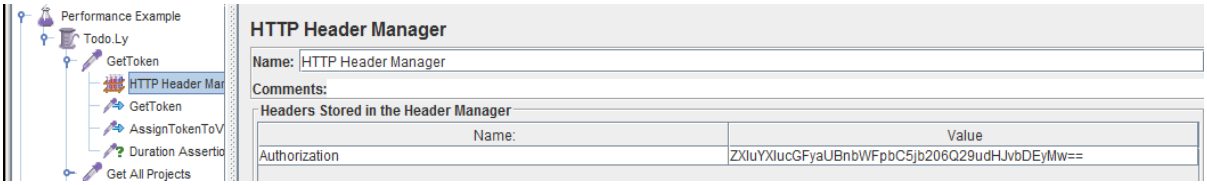
- **The configuration obj**: in this site we can save all the repetitive variables



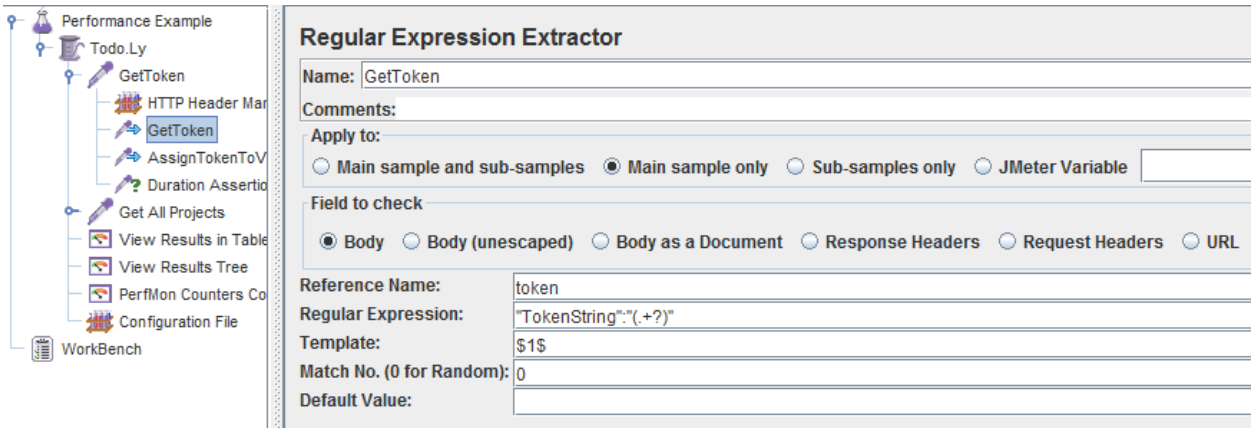
- **Get Token – End Point:** in this object we have to configure the server name or IP, the protocol to use and the URL (end point), also, we have to send the header/body parameter.

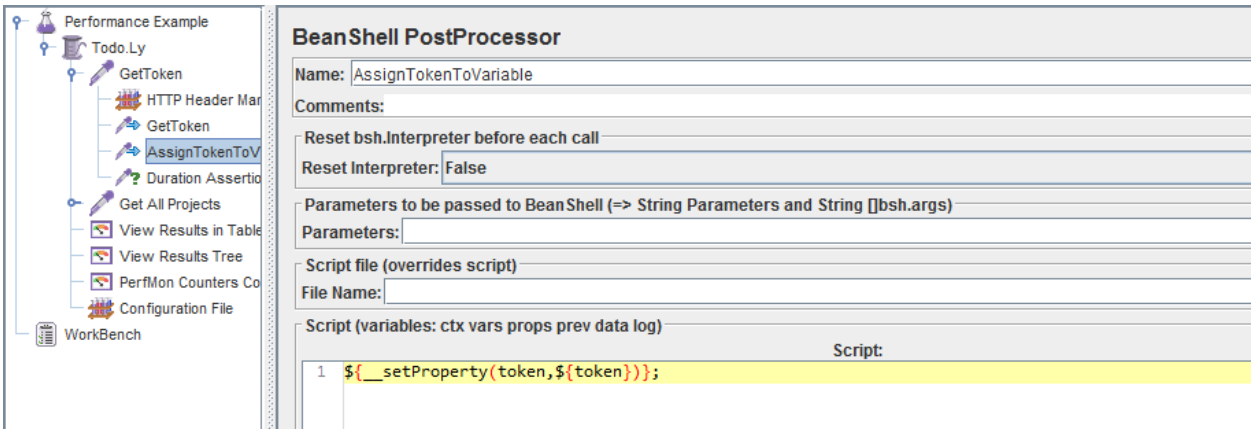


We need to send the authorization string convert to base64 (requirement of todo.ly):

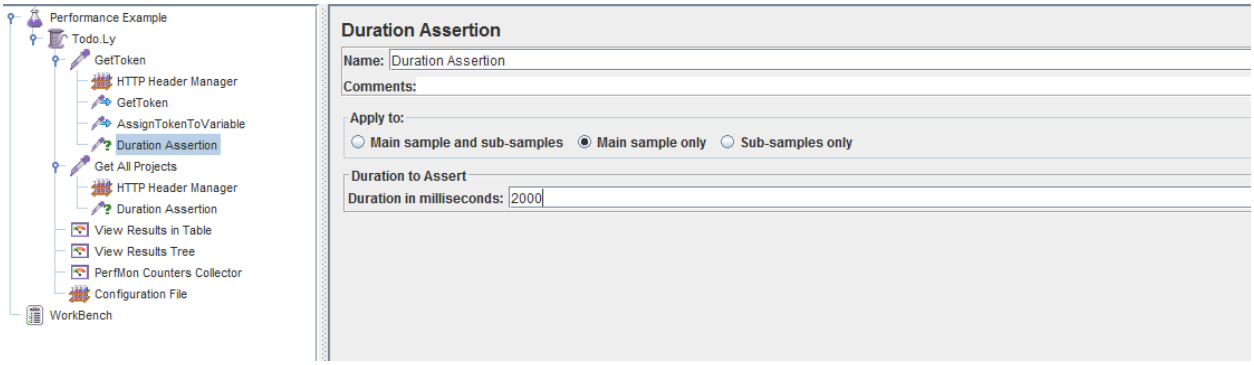


For this specific end point we need to save the token value in some variable to use in the next end points:

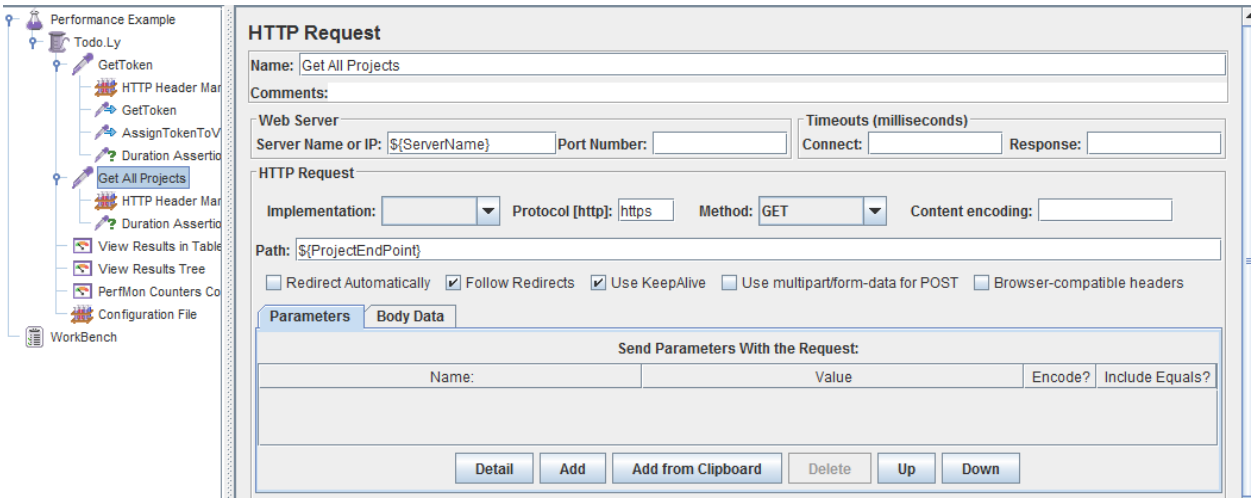




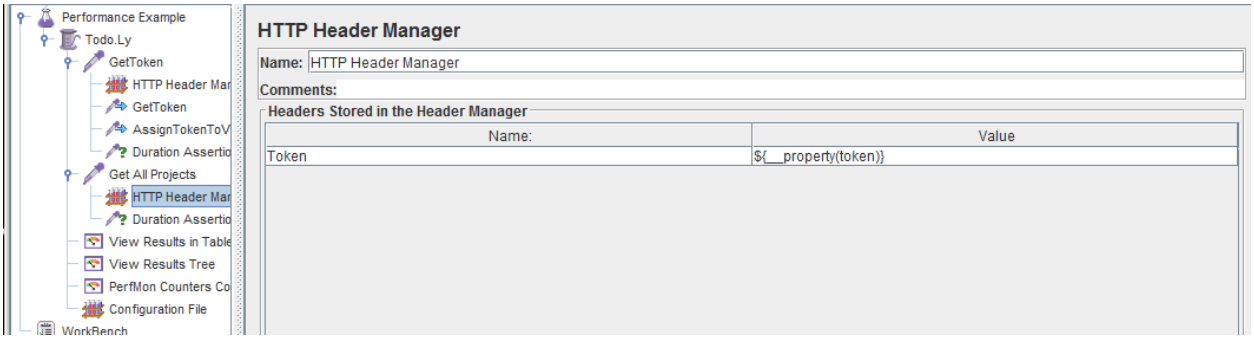
<u>Note</u>: if we have the best line time for the response time, we can add assertion using this time. Also we can add some assertion to verify the "json" retrieved or the size of bytes, etc.



- **Get All Project – End Point:** in this object we have to configure the server name or IP, the protocol to use and the URL (end point), also we have to send the header/body parameter

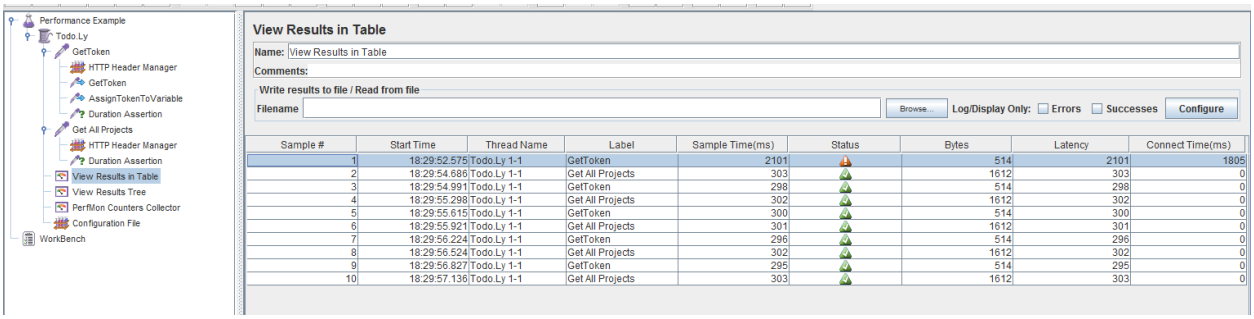


We send the token get in the previous request:

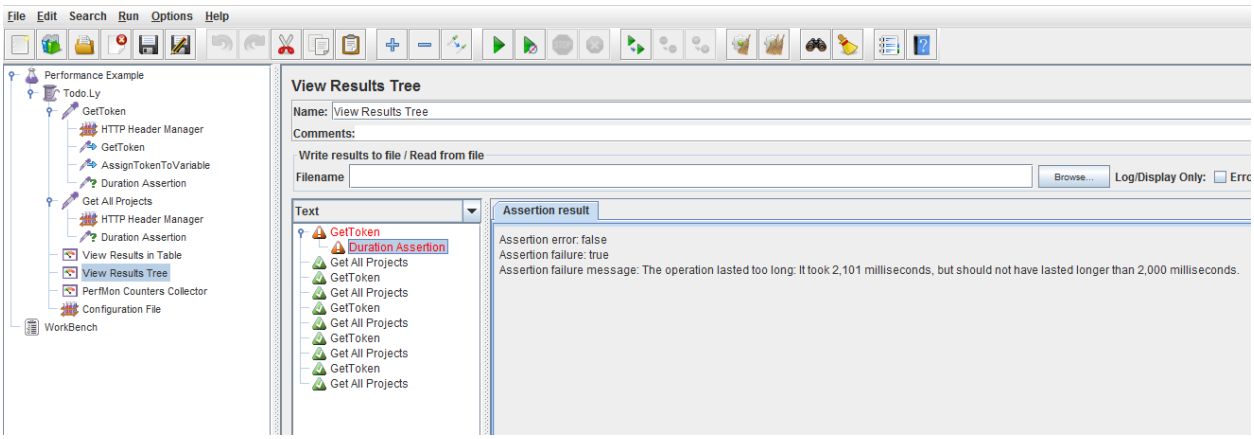


Also we add some listeners to save the result: View Result in Table, view Results Tree, Perfmon counters.

- **Results:** in this view we can see the Bytes, Latency, Sample time(response time) and the status of each end point

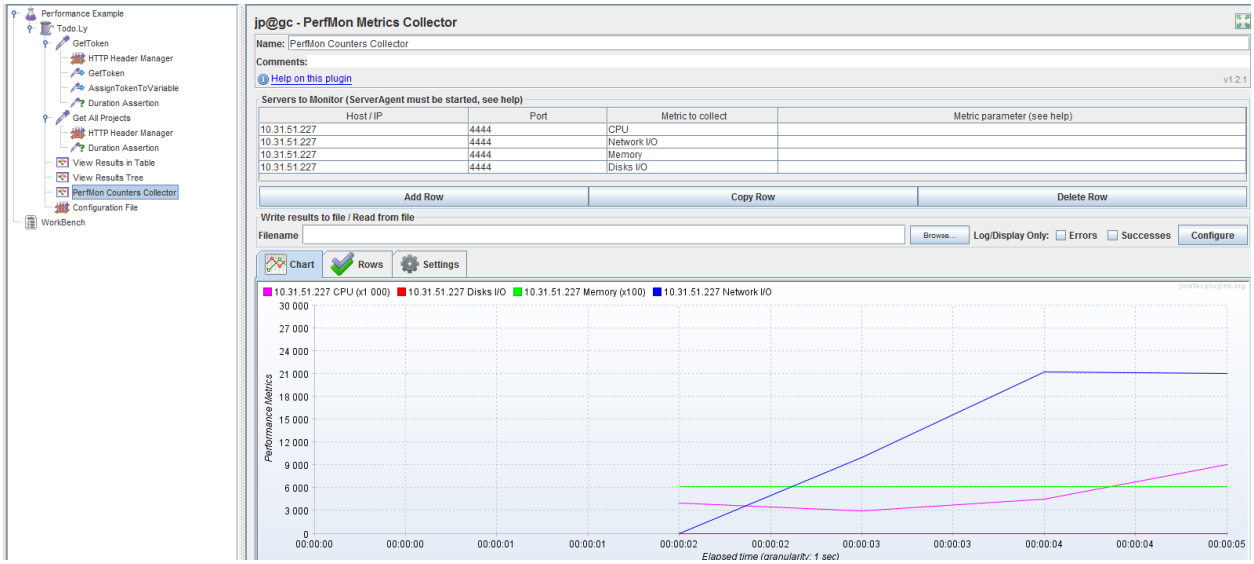


in this view we can see the details of the request, if anyone failed we can see the assertion for each one:

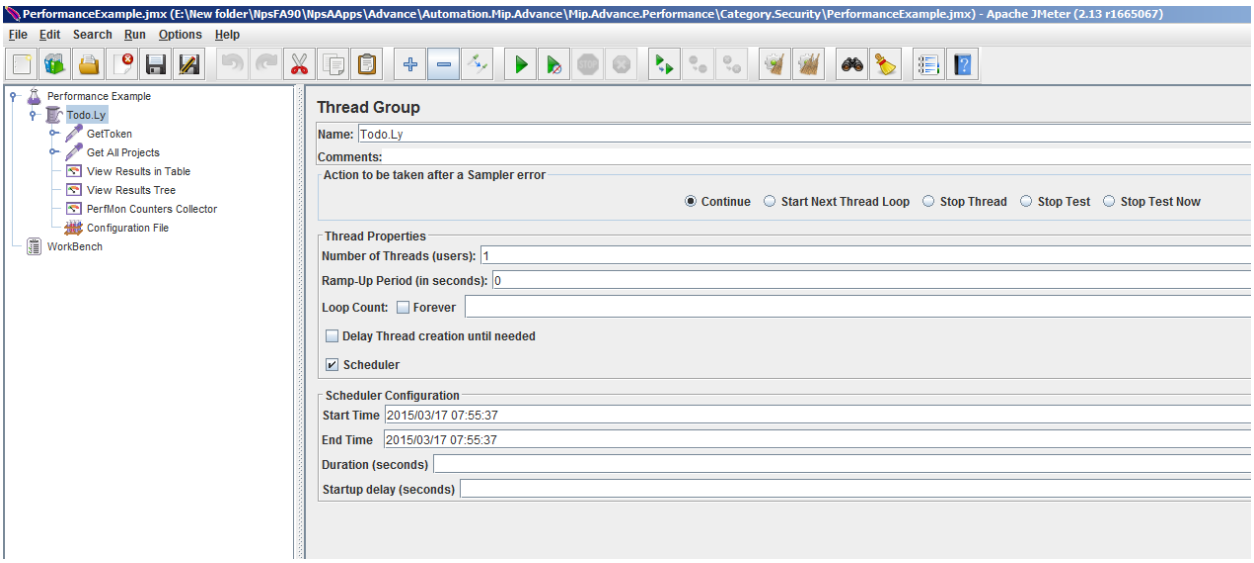


In this view we can see the behavior in the counter when the requests were executed. (Note: for use this view we need to install the agent in the server site and configure the Host/IP.),

In this case we can use other machine to monitoring it because we don't have access to Todo.ly server. Also we can use the perfmon tool to collect the counter values when we execute the request we can configure it with PowerShell or manually. Then we can get the chart on excel to analyze the data.



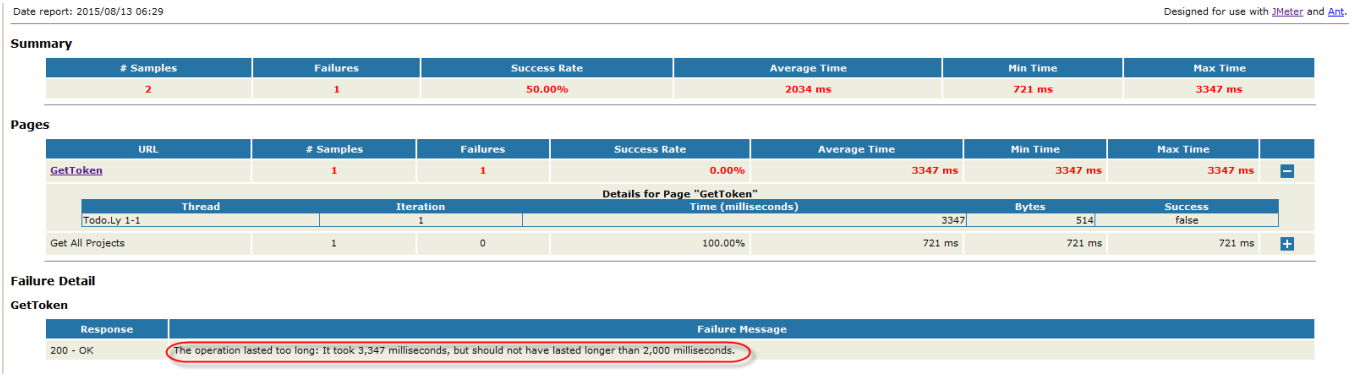
- **Execution:** We can execute all the script with the scheduler configuration, define the start time and end time (recommendation is one day to collect data), but if we need to execute just once and just compare the response time, then we define the loop Count equal to 1.



- **Ant task report:** We can use the ANT task to generate HTML report with the same information of: View Result in Table, View Result Tree and a summary.

```
C:\apache-jmeter-2.13\extras>ant -Dtest=PerformanceExample
Unable to locate tools.jar. Expected to find it in C:\Program
0_45\lib\tools.jar
Buildfile: C:\apache-jmeter-2.13\extras\build.xml
```

Note: The ant task to create the html file is configurable.



Date report: 2015/08/13 06:29 — Designed for use with JMeter and Ant.

Summary

# Samples	Failures	Success Rate	Average Time	Min Time	Max Time
2	1	50.00%	2034 ms	721 ms	3347 ms

Pages

URL	# Samples	Failures	Success Rate	Average Time	Min Time	Max Time
GetToken	1	1	0.00%	3347 ms	3347 ms	3347 ms
Get All Projects	1	0	100.00%	721 ms	721 ms	721 ms

Details for Page "GetToken"

Thread	Iteration	Time (milliseconds)	Bytes	Success
Todo.Ly 1-1	1	3347	514	false

Failure Detail

GetToken

Response	Failure Message
200 - OK	The operation lasted too long: It took 3,347 milliseconds, but should not have lasted longer than 2,000 milliseconds.

In this case the base line for this end point was 2 seconds but it was taking 3.3 seconds according to the execution results.

v. REFERENCES

- http://jmeter.apache.org/
- http://www.tutorialspoint.com/jmeter/jmeter_overview.htm
- http://artoftesting.com/performanceTesting/restAPIJMeter.html
- http://www.testautomationguru.com/how-to-test-rest-api-using-jmeter/

e. CUCUMBER (BDD)

i. ABOUT THE TOOL

Cucumber is a tool for running automated tests written in plain language that easily is adapted to BDD notation (given-when-then). It is a tool that executes plain-text functional descriptions as automated tests.

Cucumber itself is written in Ruby, but it can be used to "test" code written in Ruby or other languages including but not limited to Java, C# and Python. Cucumber only requires minimal use of Ruby programming.[K-1]

This tool is easy to use, this is a command-line tool. When execute it, it reads in the specifications from plain-language text files called features, examines them according the scenarios to test, and runs the scenarios against the specific application.

Each scenario is a list of steps for Cucumber to work through. So that Cucumber can understand these feature files, they must follow some basic syntax rules regarding BDD notation:

A. Role-feature-reason

The "role-feature-reason" template is one of the most commonly recommended aids for teams and product owners starting to write user stories:

As a
I want
So that

For example:

As a *bank customer*
I want to *withdraw money from an ATM*
So that *I am not constrained by opening hours* [K-2].

B. Given - When - Then

The Given-When-Then formula is a template intended to guide the writing of acceptance tests for a User Story:

Given *some conditions*
When *some action is carried out (Behavior)*
Then *a particular set of observable consequences of the behavior should obtain*

For example:

Given *my bank account is in credit, and I made no withdrawals recently,*
When *I attempt to withdraw an amount less than my card's limit,*
Then *the withdrawal should complete without errors or warnings*

The name for this set of rules is Gherkin. Along with the features, you give Cucumber a set of step definitions, which map the business-readable language of each step into Ruby code to carry out whatever action is being described by the step. In a mature test suite, the step definition itself will probably just be one or two lines of Ruby that delegate to a library of support code, specific to the domain of your application, that knows how to carry out common tasks on the system. Normally that will involve using an automation library, like the browser automation library Capybara, to interact with the application itself [K-3].

BDDs not also have the common and understandable language but also we have the acceptances test living in the code, so every time that any functionality is changed and the expected result is different of the previous result defined, it will be displayed as a Fail test in the result of the Cucumber execution.

In order to have cucumber running in your machine you need to:

```
Install ruby
```

Once ruby is intalled you can directly install the cucumber gems:

```
gem install cucumber
```

Note*: According the requirement of your API, you could also install any other gems like bundler, dbi, etc, to interact with other options like db conection, file transfer, etc.*

ii. EXAMPLE

The feature file should contain:

fruit.feature

```
Feature: Fruit list
In order to make a great smoothie
I need some fruit.

Scenario: List fruit
Given the system knows about the following fruit:
  | name | color |
  | banana | yellow |
```

```
  | strawberry | red |
When the client requests GET /fruits
Then the response should be JSON:
  """
  [
   {"name": "banana", "color": "yellow"},
   {"name": "strawberry", "color": "red"}
  ]
  """
```

The step definition file should look like this:

steps_fruit.rb

```
Given /^the system knows about the following fruit:$/ do |fruits|
        File.open('fruits.json', 'w') do |io|
                io.write(fruits.hashes.to_json)
        end
end

When /^the client requests GET (.*)$/ do |path|
        get(path)
end

Then /^the response should be JSON:$/ do |json|
        JSON.parse(last_response.body).should == JSON.parse(json)
end
```

Note: For the last step in order to have JSON methods, the gem json need to be installed.

iii. REFERENCES

[K-1]https://github.com/cucumber/cucumber/wiki
[k-2] http://guide.agilealliance.org/guide/rolefeature.html
[K-3] Matt Wynne and Aslak Hellesoy, "The cucumber book" The Pragmatic Bookshelf Dallas, Texas • Raleigh, North Carolina, pp. 7.

f. WIRESHARK

i. ABOUT THE TOOL

Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

The following are some of the many features Wireshark provides:

- Available for UNIX, Windows and MAC.
- Capture live packet data from a network interface.
- Open files containing packet data captured with tcpdump/WinDump, Wireshark, and a number of other packet capture programs.
- Import packets from text files containing hex dumps of packet data.
- Display packets with very detailed protocol information.
- Save packet data captured.

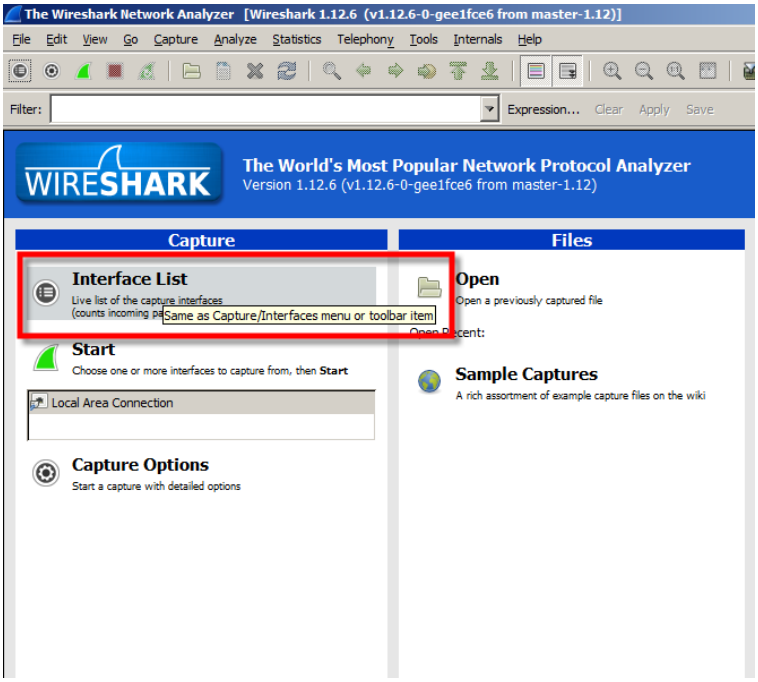
- Export some or all packets in a number of capture file formats.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.
- Create various statistics.
- …and a lot more!

ii. INSTALLATION

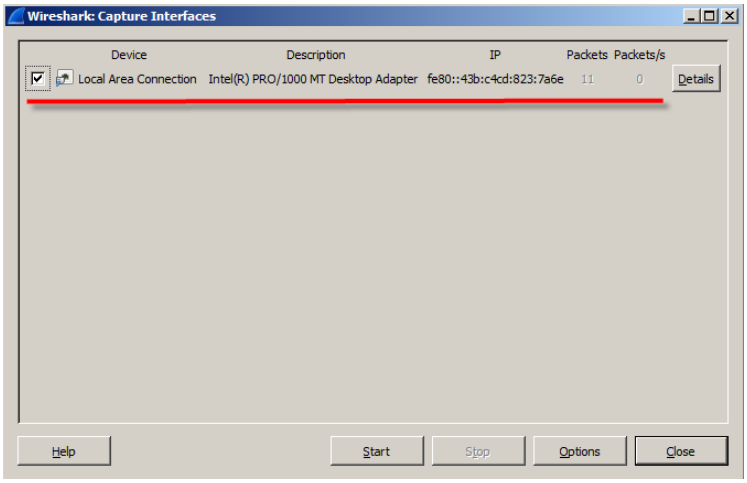
Please review the following link to install the WireShark in Windows, UNIX or MAC: https://www.wireshark.org/docs/wsug_html_chunked/ChapterBuildInstall.html

iii. EXAMPLE

1. After complete to install the WireShark, perform the following:
    a. To start to capture the traffic, first needs to select the "Interface List" that WireShark will be monitor.

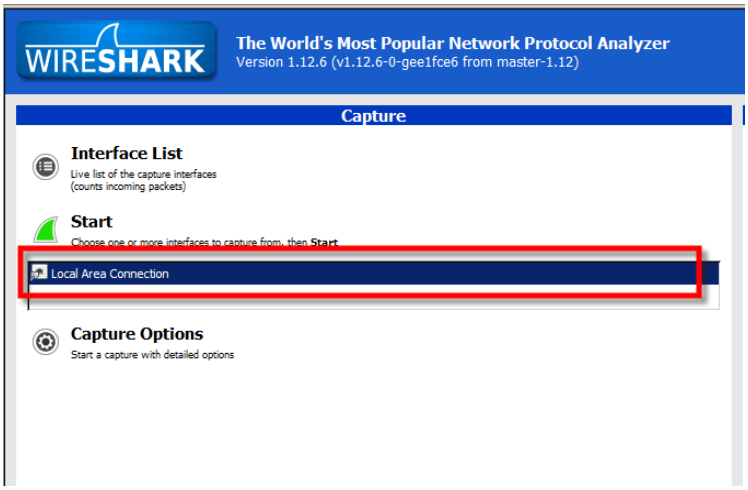


    b. A pop up with the list of Network Connections available in your Server will be opened. In this case just one Network Connection exists, but maybe you will have more than one connection, for example Bluetooth Network Connection, VMWare Network Adapter VMNetxxx. Please select the only the interfaces that you need to monitor



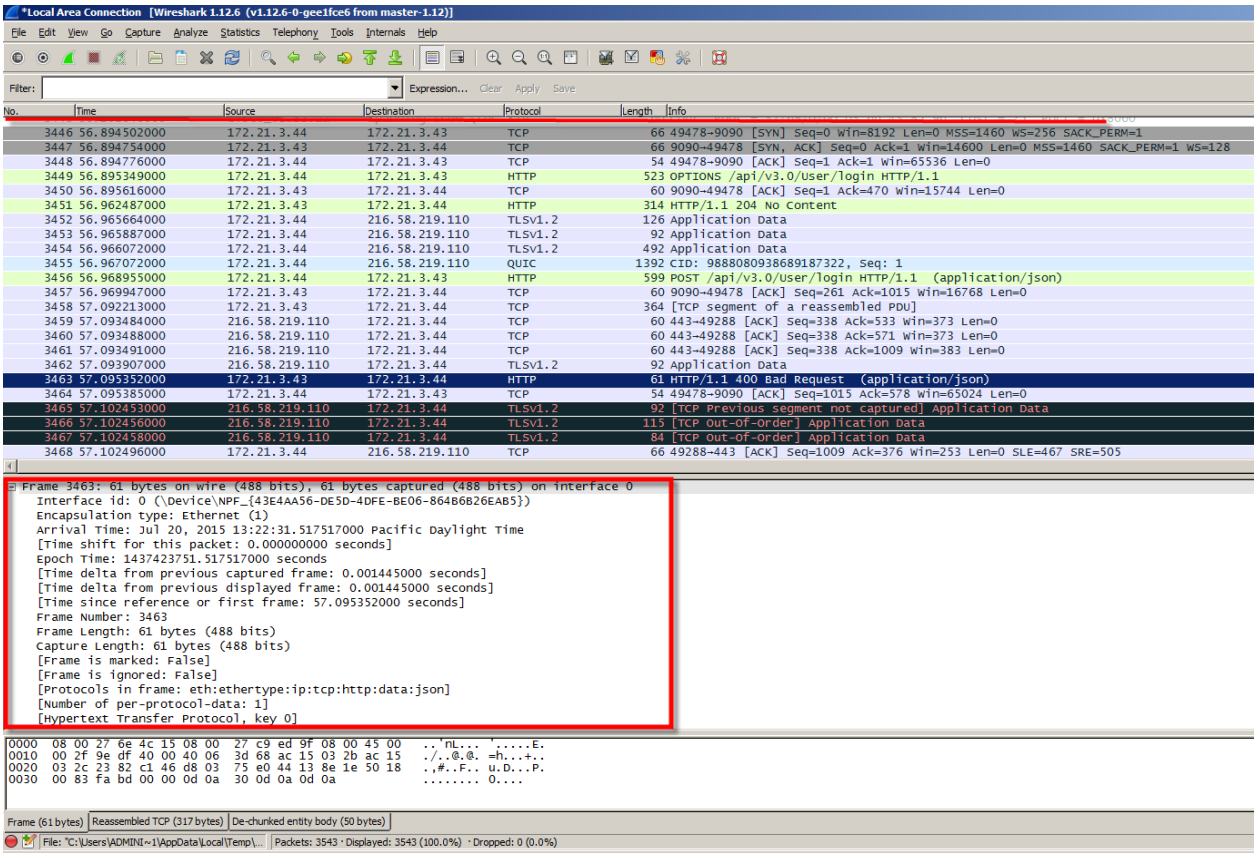
c. Select the Local Area Connection and Press Close

d. Select the Interface that you want to monitor and press Start



e. Request the URL that you need to monitor; you will see the requests and responses in the Main Windows.

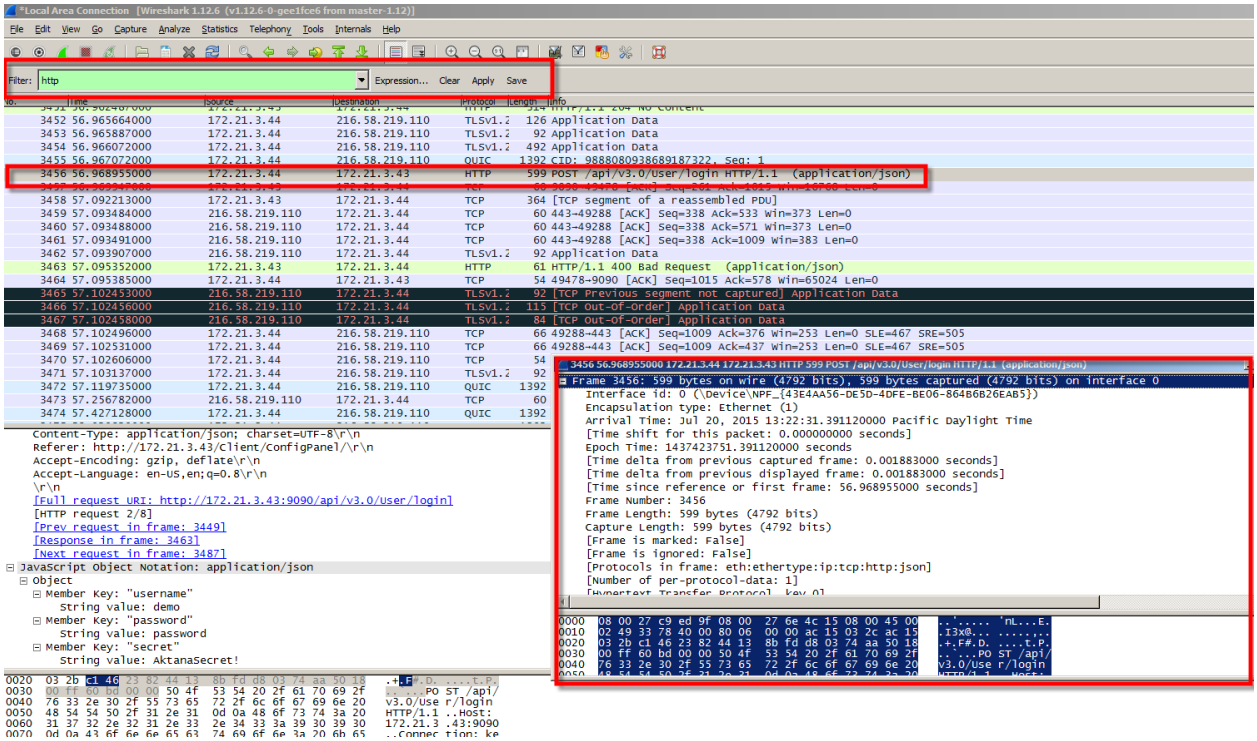
The Main Windows contains 3 panels; Packet List, Packet Detail and Packet Bytes. In the Packet List you will see the information about the all requests and response, the transactions, the time, source, destination, type of the protocol, the Length and more information. You can review the "View" Menu to setting the Main Console



f. After complete the Request that you was looking, stop to monitor. Go to \Capture\Stop

g. Apply a Filter, for this example we will only review the “http”. More information about how to use more filters you can find at https://wiki.wireshark.org/CaptureFilters

h. Analyze the data obtained: You can do it reviewing the data collected in the “Packet Detail” or open a new dialog box doing double Click on the request that you want to review



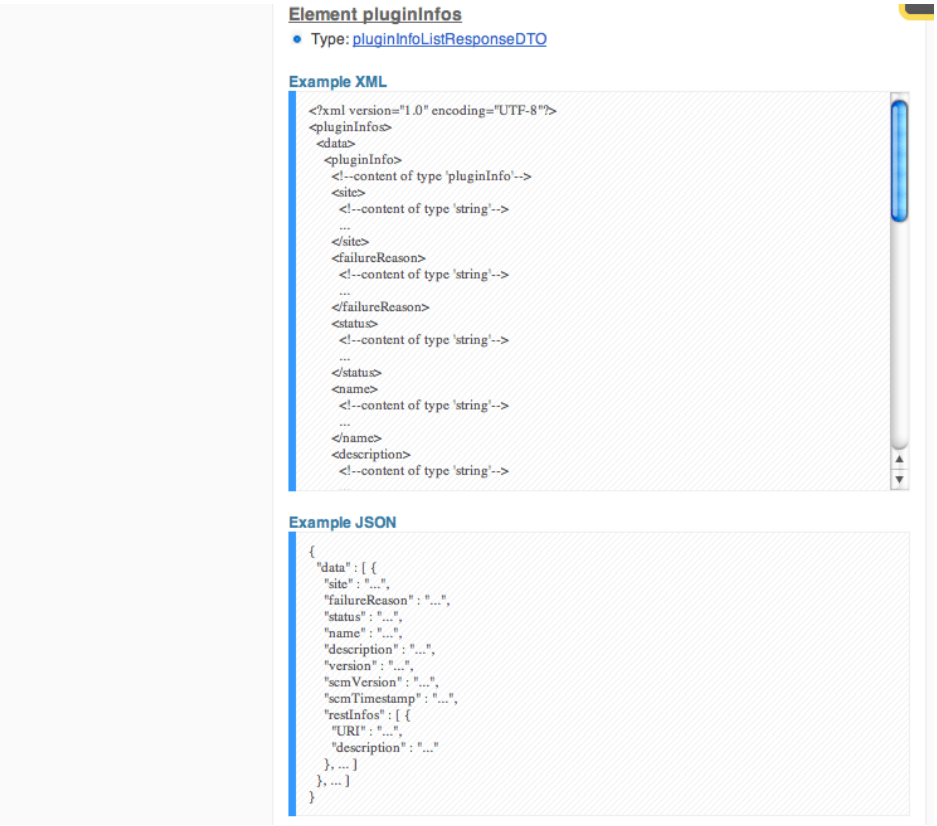
iv. REFERENCE:

- https://www.youtube.com/watch?v=qzonPrKNhwc
- https://www.wireshark.org/
- https://www.youtube.com/watch?v=FaoheEdkqdc
- To monitor SSL, please review https://wiki.wireshark.org/SSL

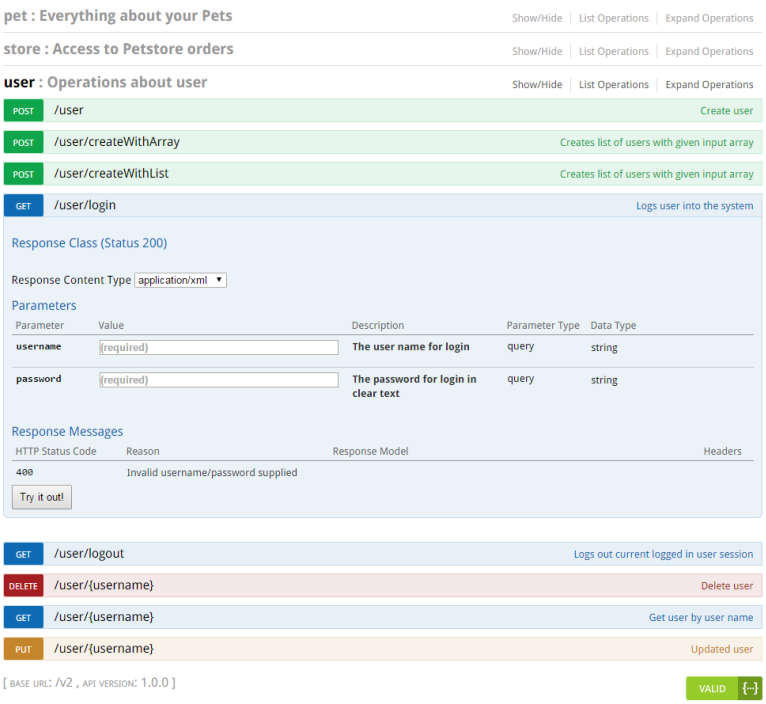
3. API REFERENCE GENERATORS

From the development side, there are some tools that can generate documentation for an API (API Reference), here are some examples:

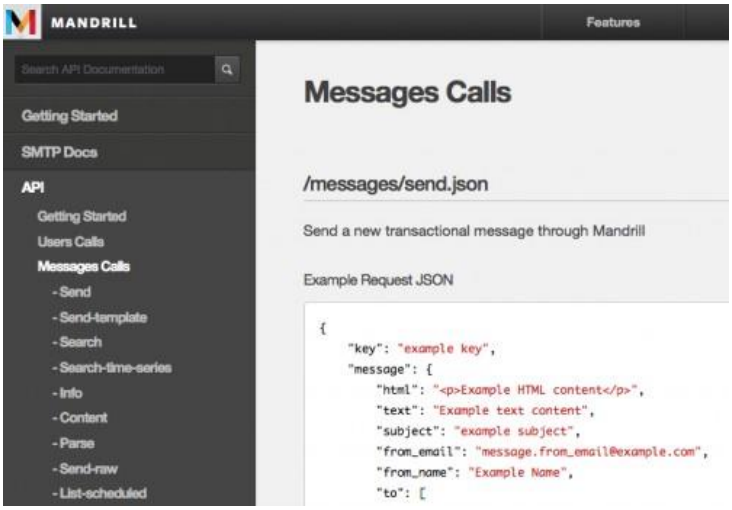
i. Enunciate: https://github.com/stoicflame/enunciate/wiki



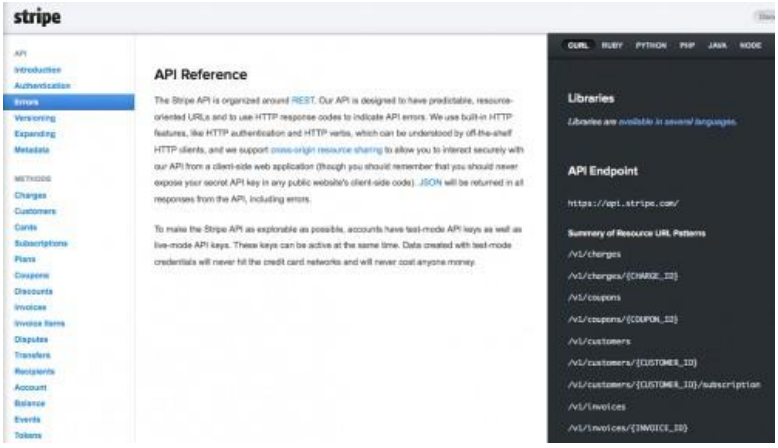
ii. Swagger: http://swagger.io



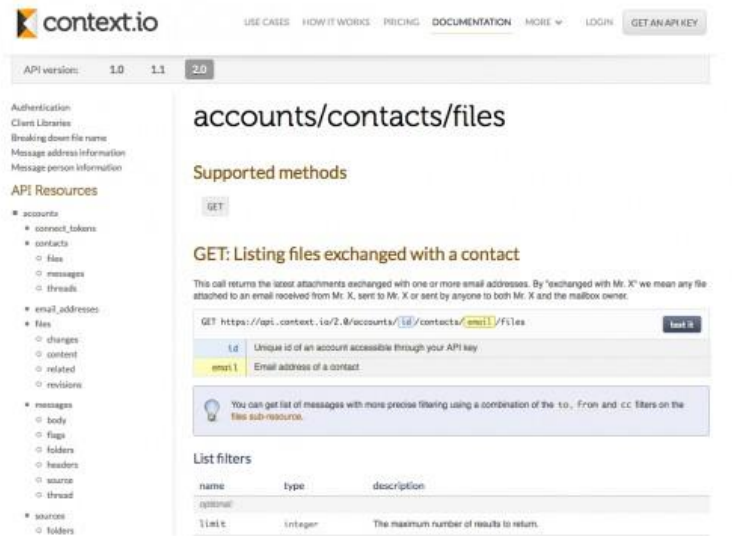
iii. Mandrill: http://mandrillapp.com



iv. Stripe: https://stripe.com/



v. Context.io: http://context.io



vi. SpringDoclet

vii. RESTdoclet
viii. Wsdoc
ix. Wadl
x. JSONdoc