



Katalon Studio For Automated Testing

Sesi 9



API TESTING⁺



How to do API Testing with Katalon

API Testing is one of the hottest trends of Software Testing in recent years and still keeps growing up till now. Instead of being handled solely by developers, testing API is presently a standard practice among many outsourcing teams. This short article will give you a comprehensive tutorial on how to perform **REST API & WebServices Testing with Katalon Studio**. Subsequently, in this article REST API & WebServices Testing

What is API testing?

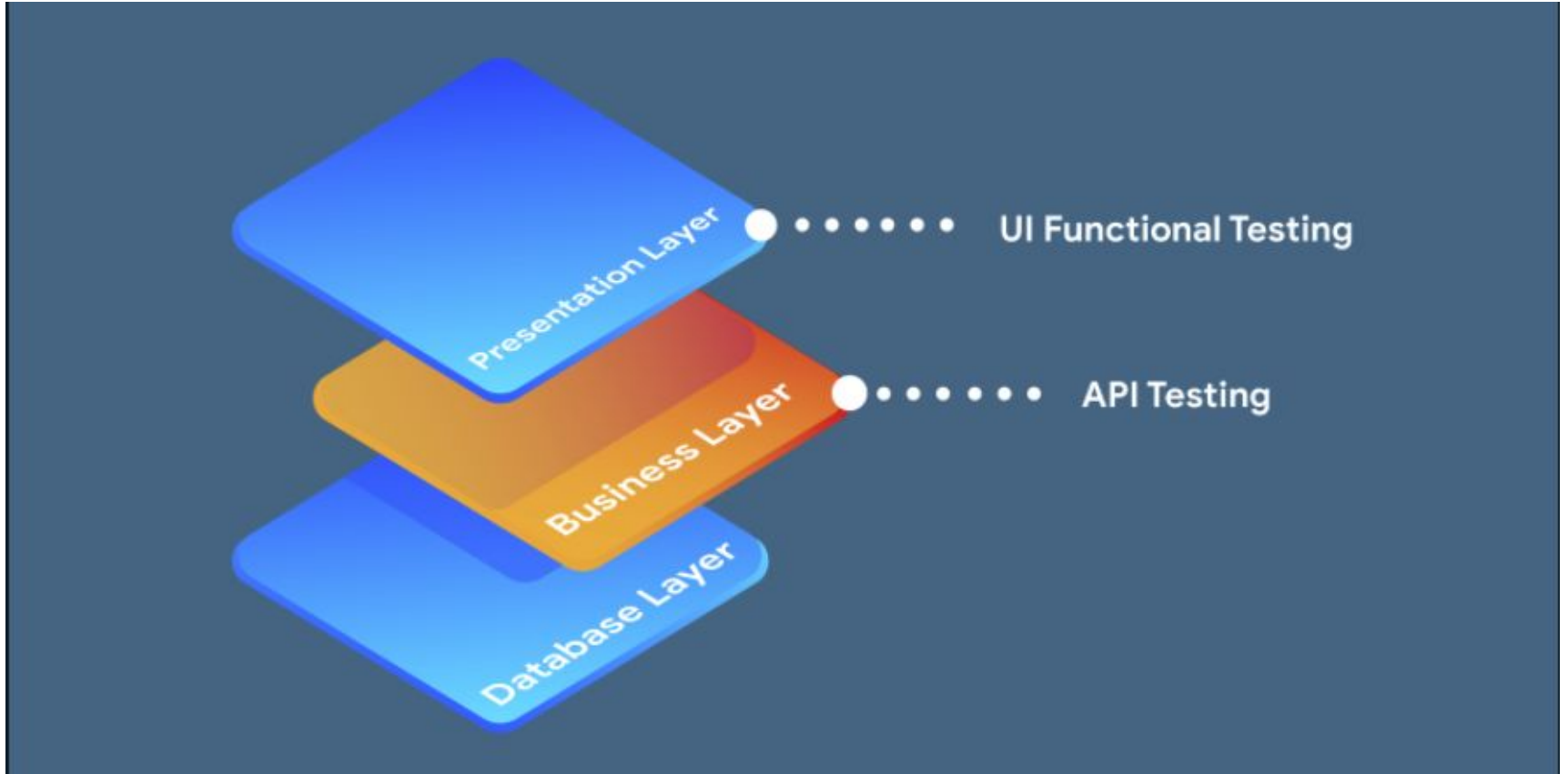
API testing is a type of software testing that involves testing application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security.

Where is API testing performed ?

Commonly, applications have three separate layers or tiers including Presentation Layer or user interface, Business Layer or application user interface for business logic processing, and Database Layer for modeling and manipulating data.

API Testing is performed at the most critical layer, the Business Layer, where business logic processing is carried out, and all transactions between User Interface and Database happen.

How to do API Testing with Katalon

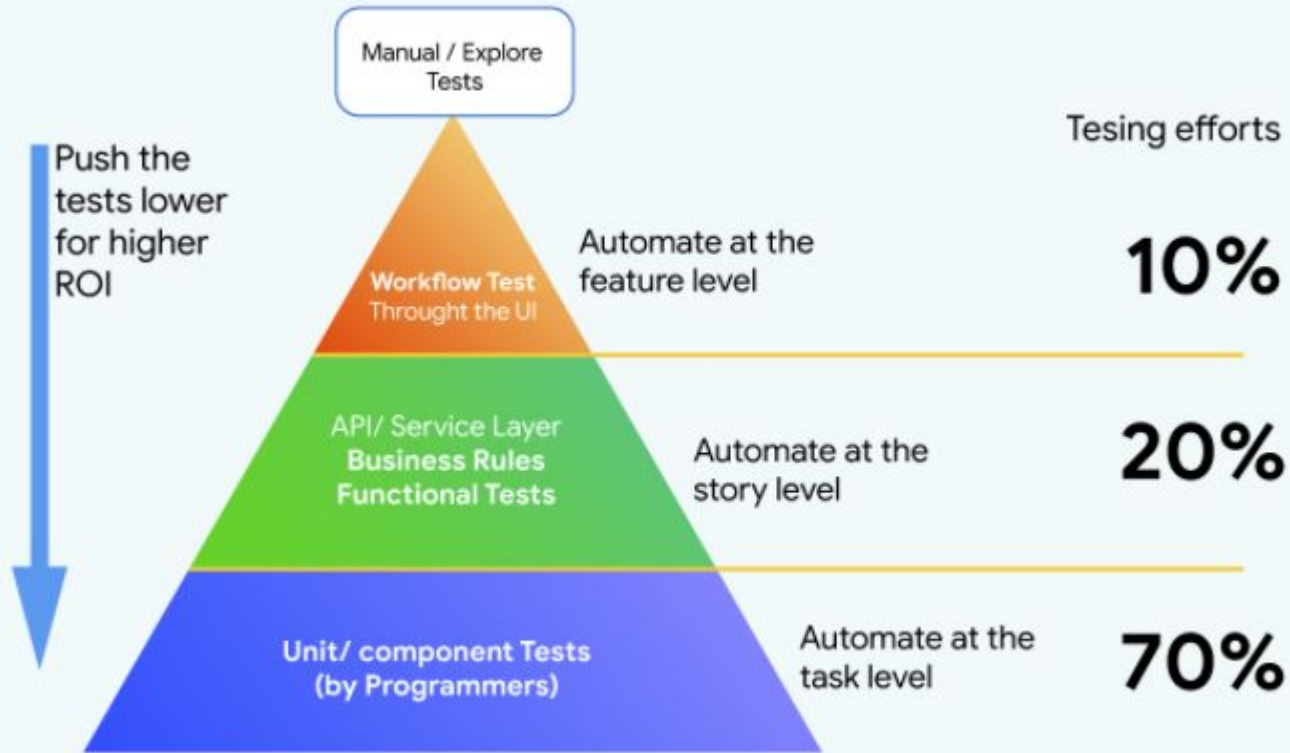


How to do API Testing with Katalon

Commonly, applications have three separate layers or tiers including Presentation Layer or user interface, Business Layer or application user interface for business logic processing, and Database Layer for modeling and manipulating data.

API Testing is performed at the most critical layer, the Business Layer, where business logic processing is carried out, and all transactions between User Interface and Database happen.

How to do API Testing with Katalon



How to do API Testing with Katalon

The figure below shows three different layers of testing called the test pyramid initially coined by Mike Cohn in his book Succeeding with Agile. It has layers representing different types of testing. Despite its being overly simplistic, it offers us a general rule of thumb: it suggests how much testing we should focus on at each layer. As such, API and services tests in the second layer is an important testing activity that we should focus on.

API TESTING TYPES

API testing types



Functional Testing



UI Testing



Load Testing



Runtime/Error Testing



Security Testing



Penetration Testing

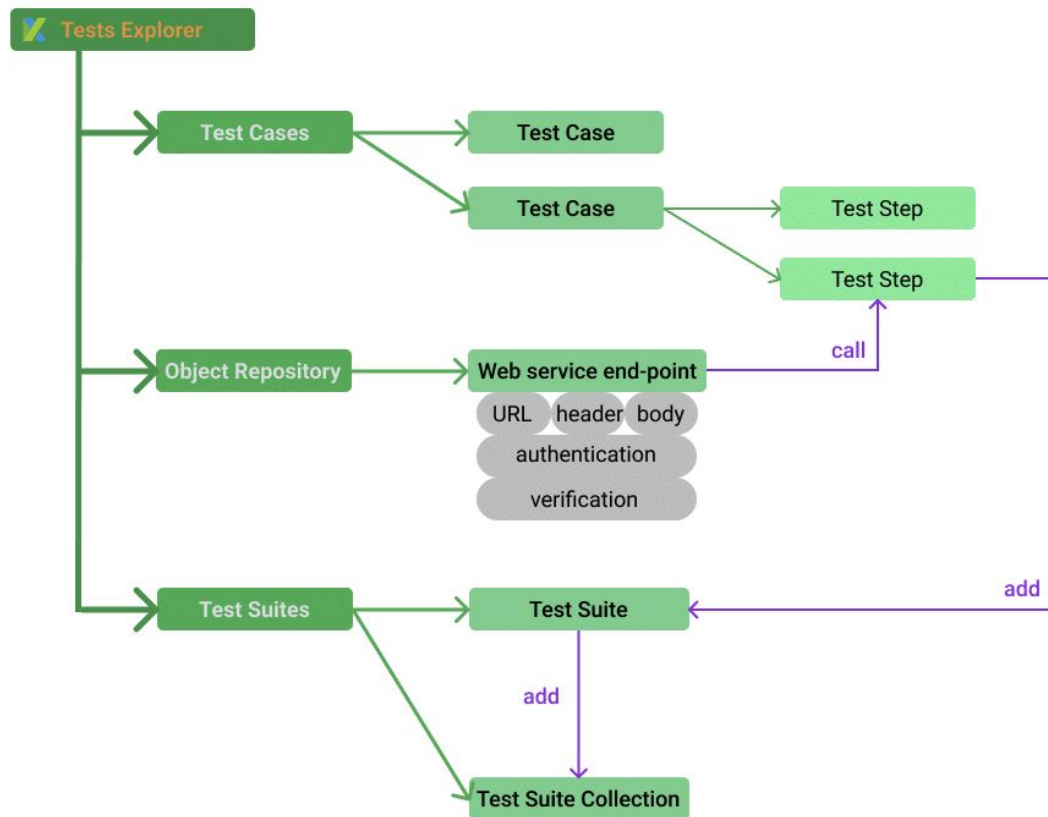


Fuzz Testing



Validation Testing

API AUTOMATION TEST



API AUTOMATION TEST

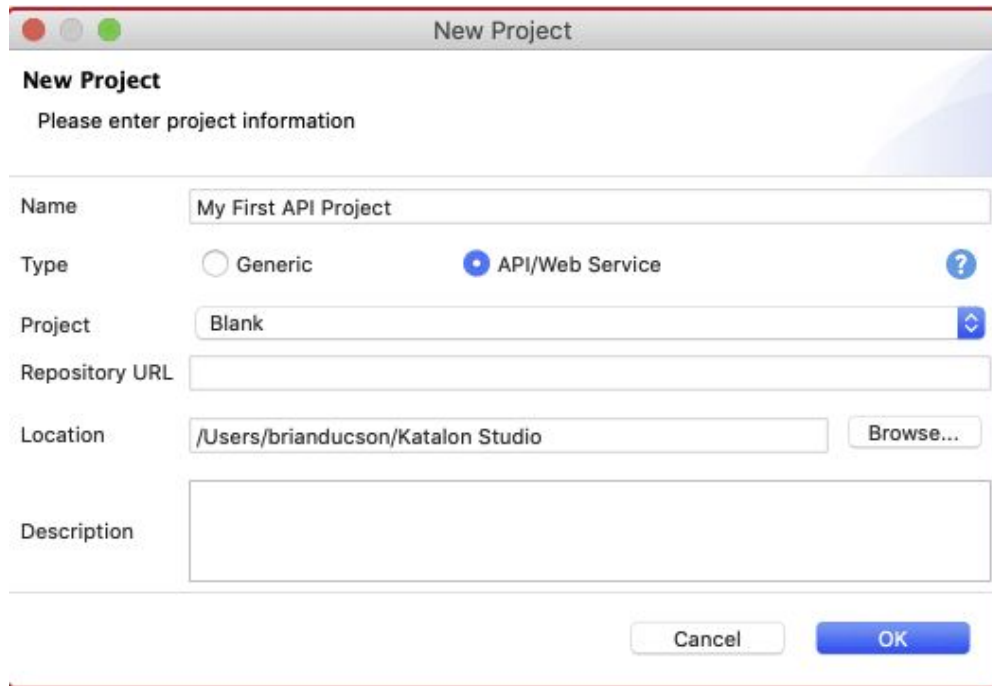
Before the creation of your first API test, let's have a look at the format used to set up a testing project.

- **Object Repository:** *Object Repository is a place that stores all Web service endpoints along with all information of request method, URL, header, content, and authentication. A folder system for better management integrates web service test objects in Object Repository.*
- **Test Cases:** *Test Cases stores/keep all test scenarios and is categorized by a folder system. Each test case includes some steps that illustrate a test scenario. You can execute a test case individually with a specified execution profile.*
- **Test Suites:** *Test Suites is where all test suites are stored. A test suite is a collection of test cases that verify a specific target. Test cases at the 'test suite' level can execute with a data-driven approach. The Test reports also generate at the 'test suite' level.*
- **Test Suite Collection:** *Test Suite Collection is a collection of Test Suites that verifies/ confirms a larger target. Test Suites at 'Test Suite' Collection level has specific Test environments specified.*

API AUTOMATION TEST

Step 1: Create a new project

Go to **File → New → Project**. Enter a project name along with its location to start a new project.



The screenshot shows the 'New Project' dialog box in Katalon Studio. The title bar reads 'New Project'. Below the title bar, the text 'New Project' is followed by 'Please enter project information'. The dialog contains several input fields and a list of project types.

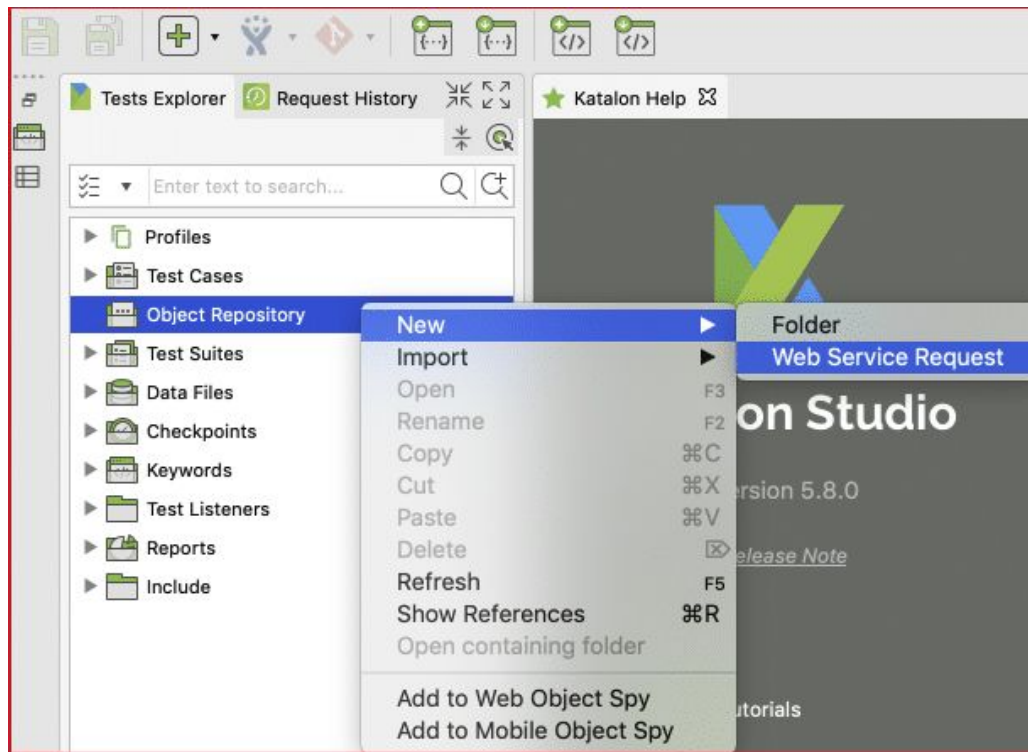
| Field | Value |
|----------------|--|
| Name | My First API Project |
| Type | <input checked="" type="radio"/> API/Web Service |
| Project | Blank |
| Repository URL | |
| Location | /Users/brianducson/Katalon Studio |
| Description | |

At the bottom of the dialog, there are two buttons: 'Cancel' and 'OK'.

API AUTOMATION TEST

Step 2: Create a new RESTful endpoint at Object Repository

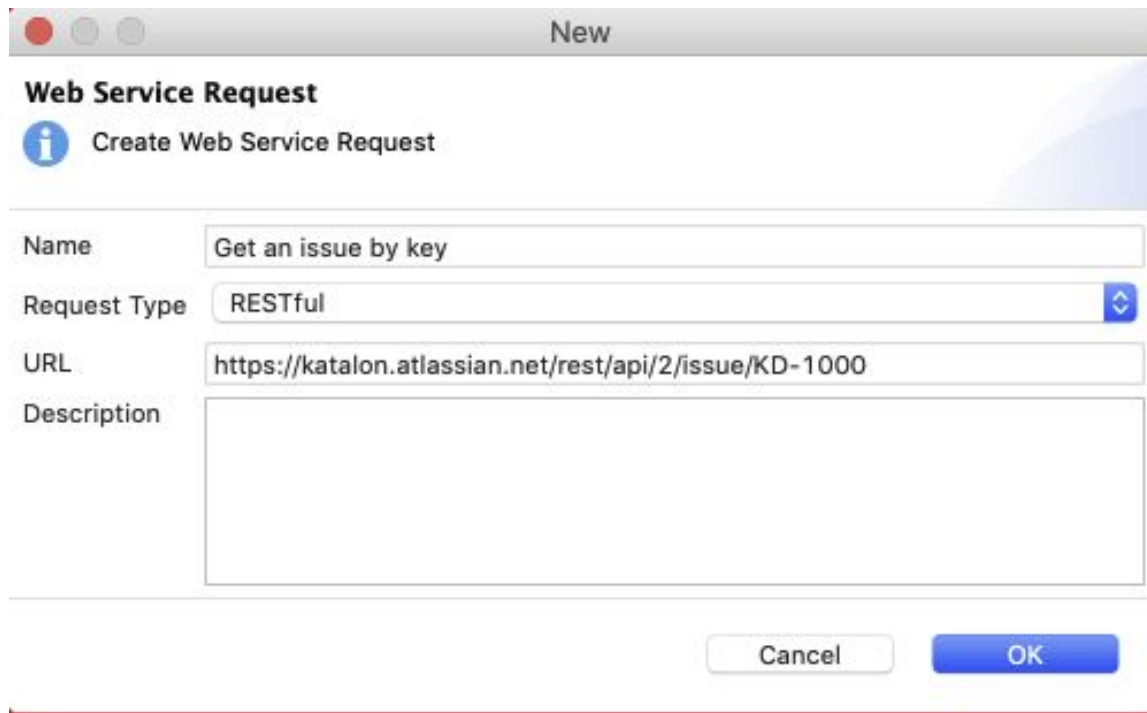
Object Repository → New → Web Service Request



API AUTOMATION TEST

Step 2: Create a new RESTful endpoint at Object Repository

Object Repository → New → Web Service Request



The screenshot shows a 'New' dialog box titled 'Web Service Request'. It contains an information icon and the text 'Create Web Service Request'. Below this, there are four input fields: 'Name' with the value 'Get an issue by key', 'Request Type' with a dropdown menu showing 'RESTful', 'URL' with the value 'https://katalon.atlassian.net/rest/api/2/issue/KD-1000', and 'Description' which is an empty text area. At the bottom right, there are 'Cancel' and 'OK' buttons.

| Field | Value |
|--------------|--|
| Name | Get an issue by key |
| Request Type | RESTful |
| URL | https://katalon.atlassian.net/rest/api/2/issue/KD-1000 |
| Description | |

API AUTOMATION TEST

Katalon Studio stores Web service endpoints for testing at Object Repository, which is similar to Test Object in UI Test. At the “*Create New Web Service Request*” dialog, you can either choose to create a RESTful or a SOAP request.

“*Request type*” is a required field. You need to specify it at this very step. In contrast, “*URL*” is not required. You can set this value later in the next step.

API AUTOMATION TEST

Click, “OK.” Then we are ready to enter more details to the first *RESTful* test.

The screenshot displays the Katalon Studio interface for configuring a RESTful test. The window title is "Katalon Studio - My First API Project - [Location: /Users/brianducson/Katalon Studio/My First API Project]". The interface is divided into several panels:

- Tests Explorer:** Located on the left, it shows a tree view of test assets including Profiles, Test Cases, Object Repository, Test Suites, Data Files, Checkpoints, Keywords, Test Listeners, Reports, and Include. The "Object Repository" is expanded, showing a test case named "GET Get an issue by key".
- Request History:** Located at the top of the main workspace, it shows the history of requests.
- Main Workspace:** The central area for configuring the test. It includes:
 - Method and URL:** A dropdown menu set to "GET" (marked with a red circle 1) and a text field containing the URL "https://katalon.atlassian.net/rest/api/2/issue/KD-1000" (marked with a red circle 2).
 - Query Parameters:** A table with columns "Name" and "Value" (marked with a red circle 3). It has an "Add" button and a "Remove" button.
 - Authorization:** A dropdown menu set to "No Authorization" (marked with a red circle 4). Below it is a button labeled "Update to HTTP Header".
 - HTTP Header:** A tab labeled "HTTP Header" (marked with a red circle 5) for adding headers.
 - HTTP Body:** A tab labeled "HTTP Body" for adding body content.
 - Verification:** A tab labeled "Verification" for adding verification rules.
 - Variables:** A tab labeled "Variables" for adding variables.
- Response:** Located on the right, it shows the response details. It includes fields for "Status", "Elapsed", and "Size". Below these are tabs for "Body" and "Header". The "Body" tab is selected, showing a response body with the text "1". There are radio buttons for "pretty", "raw", and "preview" (marked with a red circle 6).

API AUTOMATION TEST

There are some critical concepts needed to specify when testing a *RESTful* request:

- **Request method:** You can choose one of these following methods for your first request test: *GET, POST, PUT, or DELETE*. The method needs to match the URL to have a valid request. For instance, let's assume that your first test is a public API from the Jira Cloud version. In this case, you should choose/ select the *GET* method to receive information on an existing ticket using its ID.
- **Request URL:** Along with the request method, "request URL" tells the web server which API to utilize under test. Any mismatch between method and URL will lead to invalid request exceptions at runtime or wrong data response.
- **Authorization:** Authorization is a vital part of an API. It gets the correct data under permission (unless the data is public). Katalon Studio supports standard authentication methods:

Basic: The basic method requires username and password. Don't forget to click 'Update to HTTP Header' so that the authentication can apply to 'HTTP Header.'

API AUTOMATION TEST

Authorization HTTP Header HTTP Body Verification Variables

Type Basic

Username brian.ducson@gmail.com

Password

☐ Show Password

Update to HTTP Header

[illegible]

API AUTOMATION TEST

- **Verification:** Verification is where you define assertion to ascertain that the response will contain the expected information.

The screenshot displays the Katalon Studio interface for an API test. The top toolbar includes icons for file operations, test execution, and search. The main window is divided into several panes:

- Test Script Editor:** Contains a Groovy script for testing the API endpoint `https://katalon.atlassian.net/rest/api/2/issue/KD-1000`. The script includes assertions for the response status code and specific fields.

```
import static org.assertj.core.api.Assertions.*

RequestObject request = WSResponseManager.getInstance().getCurrentRequest()
ResponseObject response = WSResponseManager.getInstance().getCurrentResponse()

// Verify the response
WS.verifyResponseStatusCode(response, 200)

WS.verifyElementPropertyValue(response, 'fields.project.key', 'KD')
WS.verifyElementPropertyValue(response, 'fields.issueType.name', 'Task')
WS.verifyElementPropertyValue(response, 'fields.priority.name', 'Medium')
WS.verifyElementPropertyValue(response, 'fields.summary', 'REST - Create new RESTf
WS.verifyElementPropertyValue(response, 'fields.description', 'As a Katalon user,
```
- Test Request And Verify:** A button to execute the test script and verify the response.
- Response:** Displays the response details, including the elapsed time (2429 ms) and size (5 KB). It shows the response body in JSON format, which includes fields like `expand`, `id`, `self`, `key`, `fields`, `issueType`, `description`, `iconUrl`, `name`, `subtask`, `avatarId`, `timespent`, and `project`.
- SNIPPETS:** A list of reusable snippets for common API test actions, such as "Get current response", "Get current request", "Get a global variable", "Get a variable", "Response body: Contains string", "Response body: Convert to JSON Object", "Response body: Is equal to a String", "Response body: JSON value check", and "Response headers: Content-Type header".

API AUTOMATION TEST

The verification tab of a request is similar to the Script tab of a test case. In other words, you can write custom scripts with built-in keywords or Groovy/Java scripts to check the response data. Besides built-in keywords, Katalon Studio also supports built-in snippets, which help you generate assertions with a single click. This feature is useful for testers who might find it challenging to deal with parsing or to assert with JSON data format.

The right panel of the request consists of the responses automatically displayed in a friendly format, and the verification results in Verification Log. To include verification script when sending the request, you need to choose the 'Test Request and Verify' option from the execution button.

The Verification script gives you quick feedback on the request status rather than an actual test. You can add more assertions at the 'test case' level in the next step.

API AUTOMATION TEST

- **Variables:** Variables make API testing more strong and dynamic with the data-driven approach. In Katalon Studio, every part of the request can parameterize. In other words, we can use dynamic data for URL, authentication, HTTP Header, and HTTP Body to maximize the capability of data-driven testing. Let's have a look at this example:

The screenshot displays the Katalon Studio interface for an API test. The top toolbar includes icons for saving, running, and debugging. The main workspace is divided into two panes. The left pane shows the test configuration for a GET request to `https://katalon.atlassian.net/rest/api/2/issue/${issue_key}`. Below the URL, the 'Query Parameters' section is empty. The 'Variables' tab is active, showing a table with one variable:

| No. | Name | Type | Default value | Description |
|-----|-----------|--------|---------------|-------------|
| 1 | issue_key | String | "KD-1000" | |

A red arrow points from the 'issue_key' variable in the table to its usage in the URL. The right pane shows the 'Response' section with a status of '200 OK', an elapsed time of '2404 ms', and a size of '5 KB'. The response body is displayed in JSON format, showing details for an issue with ID '10999' and key 'KD-1000'.

```
{
  "expand": "renderedFields, names, schema, operations, editmeta, change1",
  "id": "10999",
  "self": "https://katalon.atlassian.net/rest/api/2/issue/10999",
  "key": "KD-1000",
  "fields": {
    "issuetype": {
      "self": "https://katalon.atlassian.net/rest/api/2/issuetype/10",
      "id": "10002",
      "description": "A task that needs to be done.",
      "iconUrl": "https://katalon.atlassian.net/secure/viewavatar?si"
    }
  }
}
```

- **Formatter:** The response will automatically display in a proper format: JSON, XML, HTML, and JavaScript. It gives you a quick view of the response status.

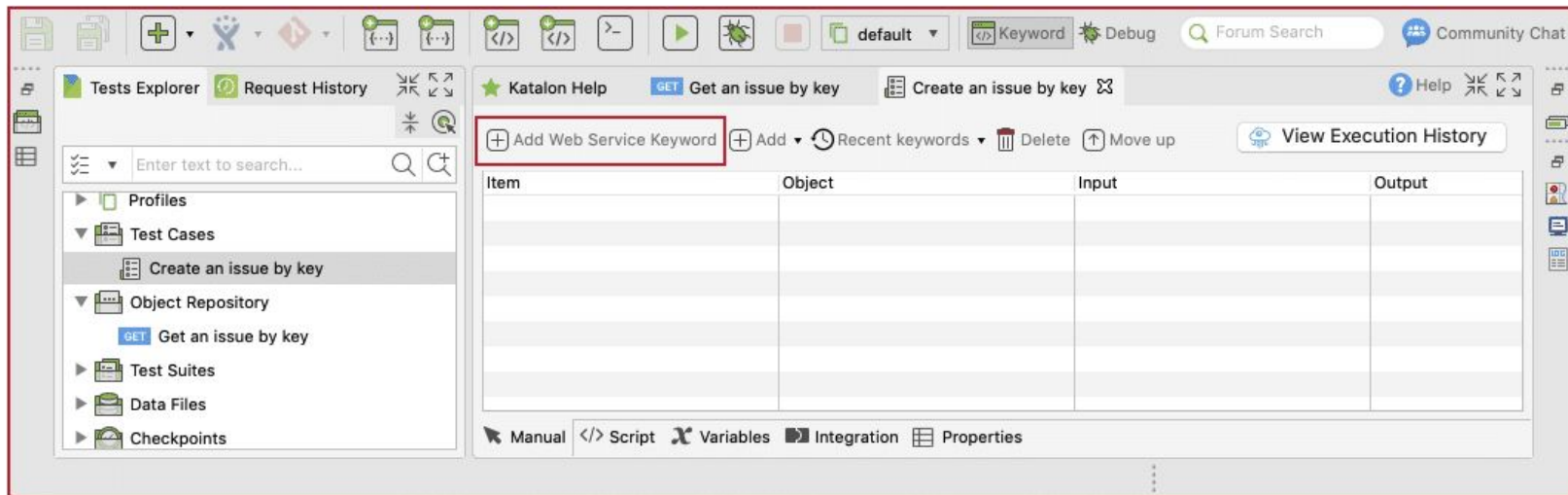
API AUTOMATION TEST

Step 3: Create a new test case with an existing request.

While the request at **Object Repository** is helpful for fast testing, you can add the request verification at the test case level for better managing and reporting.

Step 4: Add an existing request to a test case

A request can insert into a test case with Web service built-in keywords. We can use many keywords to send the request, to verify the response, and to make the request as part of a more significant testing flow.



API AUTOMATION TEST

Following test case shows how we can call the request with verification steps from a test case:

The screenshot displays the Katalon Studio interface. On the left, the 'Tests Explorer' pane shows a tree structure with 'Test Cases' expanded, and 'Create an issue by key' selected. A red arrow points from this test case to the main editor. The main editor shows a test case with the following steps:

| Item | Object | Input | Output |
|-----------------------------------|---------------------|--------------------------------------|----------|
| 1 - Send Request | Get an issue by key | | response |
| 2 - Verify Response Status Code | | response; 200 | |
| 3 - Verify Element Property Value | | response; "fields.project.key"; "KD | |
| 4 - Verify Element Property Value | | response; "fields.issuetype.name"; | |
| 5 - Verify Element Property Value | | response; "fields.priority.name"; "M | |
| 6 - Verify Element Property Value | | response; "fields.summary"; "REST | |
| 7 - Verify Element Property Value | | response; "fields.description"; "As | |

Red arrows indicate the flow from the test case in the explorer to the first step, and from the first step to the subsequent verification steps. The bottom of the interface shows tabs for 'Manual', 'Script', 'Variables', 'Integration', and 'Properties'.

API AUTOMATION TEST

The test case can execute as a standard test case in Katalon Studio. We can view each verification step from the log.

The screenshot displays the Katalon Studio interface during a test case execution. The left sidebar shows the 'Test Cases' folder with a sub-item 'Create an issue by key'. The main workspace shows a table of test steps:

| Item | Object | Input | Output |
|-----------------------------------|---------------------|--------------------------------------|----------|
| 1 - Send Request | Get an issue by key | | response |
| 2 - Verify Response Status Code | | response; 200 | |
| 3 - Verify Element Property Value | | response; "fields.project.key"; "KD | |
| 4 - Verify Element Property Value | | response; "fields.issue.type.name"; | |
| 5 - Verify Element Property Value | | response; "fields.priority.name"; "A | |
| 6 - Verify Element Property Value | | response; "fields.summary"; "REST | |
| 7 - Verify Element Property Value | | response; "fields.description"; "As | |

Below the table, the 'Console' tab shows the execution log. The log indicates that the test case 'Test Cases/Simple examples/api-2-issue/Get issue/Get an issi' was executed successfully. The log details include:

- Runs: 1/1
- Passes: 1
- Failures: 0
- Errors: 0
- Elapsed time: 3.643
- Start: 10-08-2018 05:56:08 PM
- Message: Test Cases/Simple examples/api-2-issue/Get issue/Get an issi

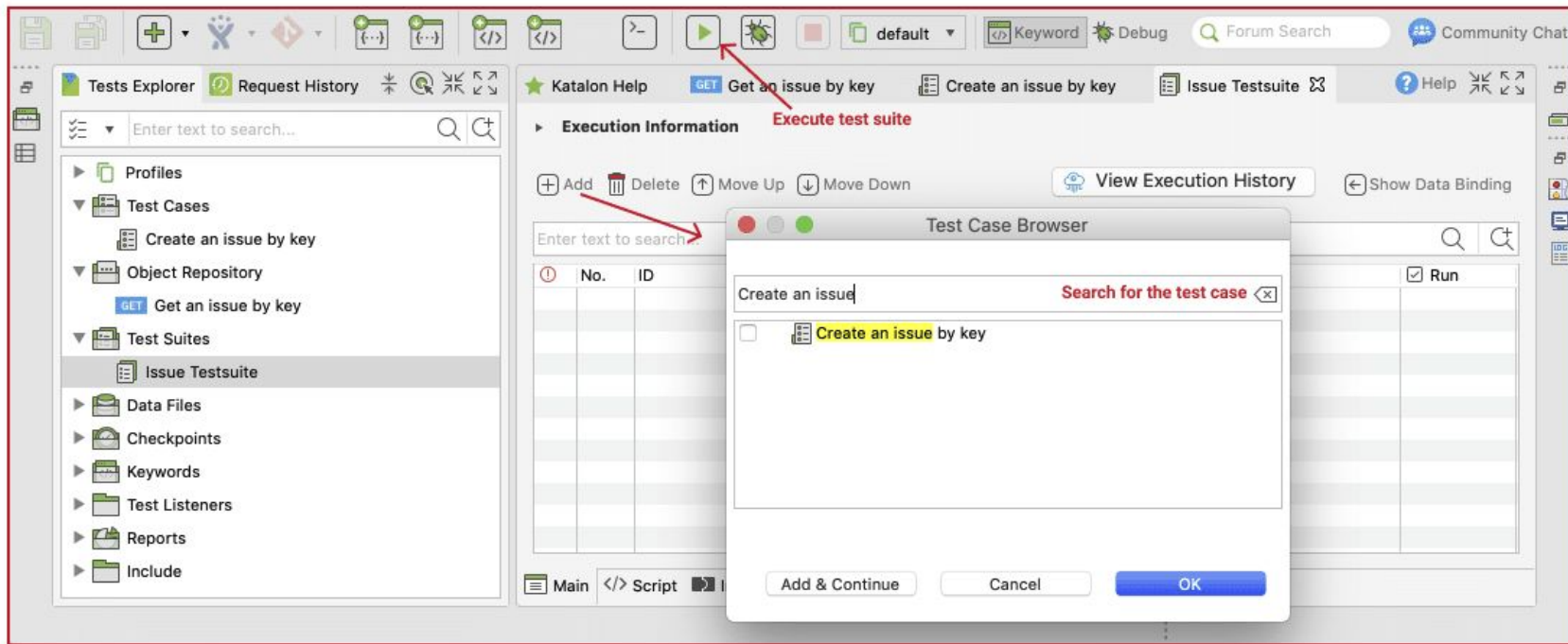
The log also shows the following steps and their durations:

- sampleBeforeTestCase (0.041s)
- 1 - Statement - response = com.kms.katalon.core.webservice.ke (0.016s)
- 2 - verifyResponseStatusCode (0.016s)
- 3 - verifyElementPropertyValue (0.081s)
- 4 - verifyElementPropertyValue (0.013s)
- 5 - verifyElementPropertyValue (0.016s)
- 6 - verifyElementPropertyValue (0.011s)
- 7 - verifyElementPropertyValue (0.017s)
- sampleAfterTestCase (0.013s)

API AUTOMATION TEST

Step 5: Add a test case to the test suite

A test case can add to a test suite via either the drag-and-drop feature or the “Add test case” tool. Once the test case adds to the test suite, we can execute the entire test suite with the Run button (*without selecting the browser as in UI testing*)



REST API

1. Buka Web Reqres.in

| | |
|------|-----------------------------|
| GET | LIST USERS |
| GET | SINGLE USER |
| GET | SINGLE USER NOT FOUND |
| GET | LIST <RESOURCE> |
| GET | SINGLE <RESOURCE> |
| GET | SINGLE <RESOURCE> NOT FOUND |
| POST | CREATE |
| PUT | UPDATE |

Request
`/api/users?page=2`

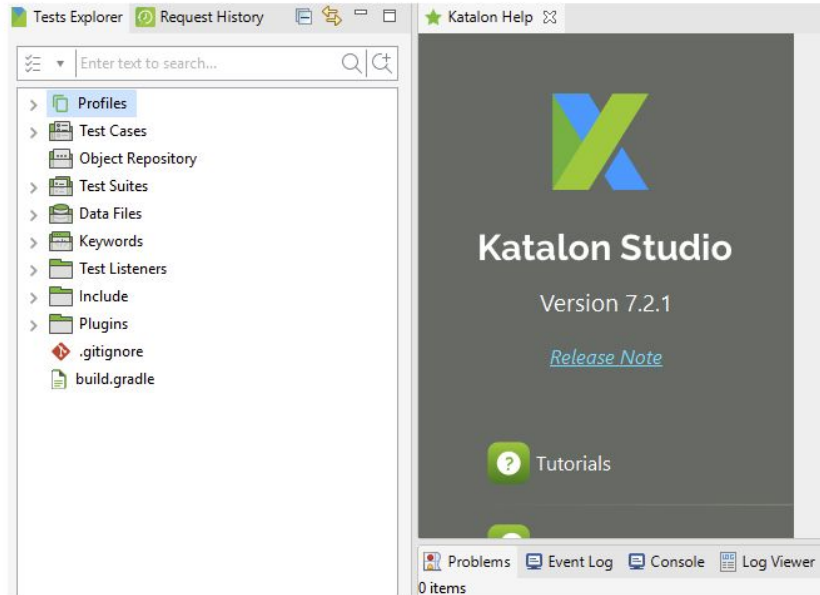
Response
200

```
{
  "page": 2,
  "per_page": 6,
  "total": 12,
  "total_pages": 2,
  "data": [
    {
      "id": 7,
      "email": "michael.lawson@reqres.",
      "first_name": "Michael",
      "last_name": "Lawson",
      "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/7/avatar.png"
    },
    {
      "id": 8,
      "email": "lindsay.ferguson@reqres.",
      "first_name": "Lindsay",
      "last_name": "Ferguson",
      "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/8/avatar.png"
    }
  ]
}
```

REST API

Pilih menu FILE > NEW > Project untuk membuat project baru. Lalu akan muncul kotak dialog yang perlu diisi sebagai data project tersebut:

- Name: isikan nama project (bebas)
- Type: isikan API/Web Service
- Lalu klik OK. Tunggu hingga loading selesai. Maka akan terlihat susunan test explorer yang terdiri dari profiles, test cases, object repository, test suites, dan lain-lain.



REST API

Konfigurasi Request

Link URL: <https://reqres.in/api/users?page=2>

Method: GET

| | |
|------|-----------------------------|
| GET | LIST USERS |
| GET | SINGLE USER |
| GET | SINGLE USER NOT FOUND |
| GET | LIST <RESOURCE> |
| GET | SINGLE <RESOURCE> |
| GET | SINGLE <RESOURCE> NOT FOUND |
| POST | CREATE |
| PUT | UPDATE |

Request
`/api/users?page=2`

Response
`200`

```
{
  "page": 2,
  "per_page": 6,
  "total": 12,
  "total_pages": 2,
  "data": [
    {
      "id": 7,
      "email": "michael.lawson@reqres.",
      "first_name": "Michael",
      "last_name": "Lawson",
      "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/7/avatar.png"
    },
    {
      "id": 8,
      "email": "lindsay.ferguson@reqres.",
      "first_name": "Lindsay",
      "last_name": "Ferguson",
      "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/8/avatar.png"
    }
  ]
}
```

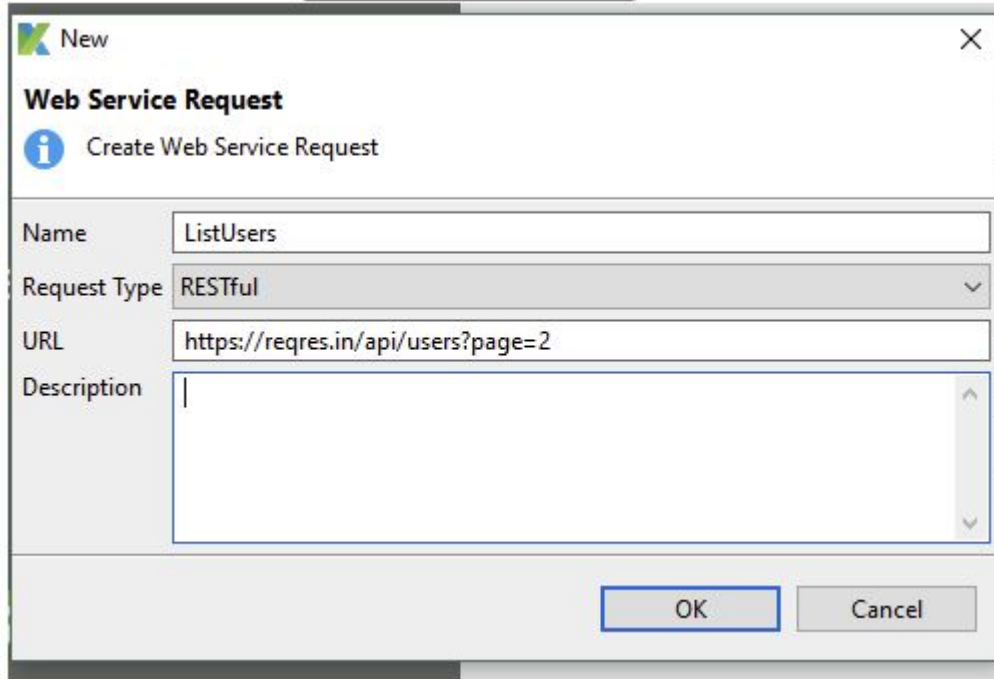
REST API

Langkah berikutnya adalah konfigurasi request. Pertama kita perlu menyusun dulu bagaimana foldering di Katalon agar rapi. Maka pada Object Repository > New > Folder. Beri nama folder tersebut REST Service.

Endpoint yang bakal kita pakai adalah pada website <https://reqres.in/> di bagian List User. URL untuk mendapatkan response tersebut menjadi,

REST API

Selanjutnya, pada folder **REST Service** tadi, ***klik kanan > NEW > Web Service Request***



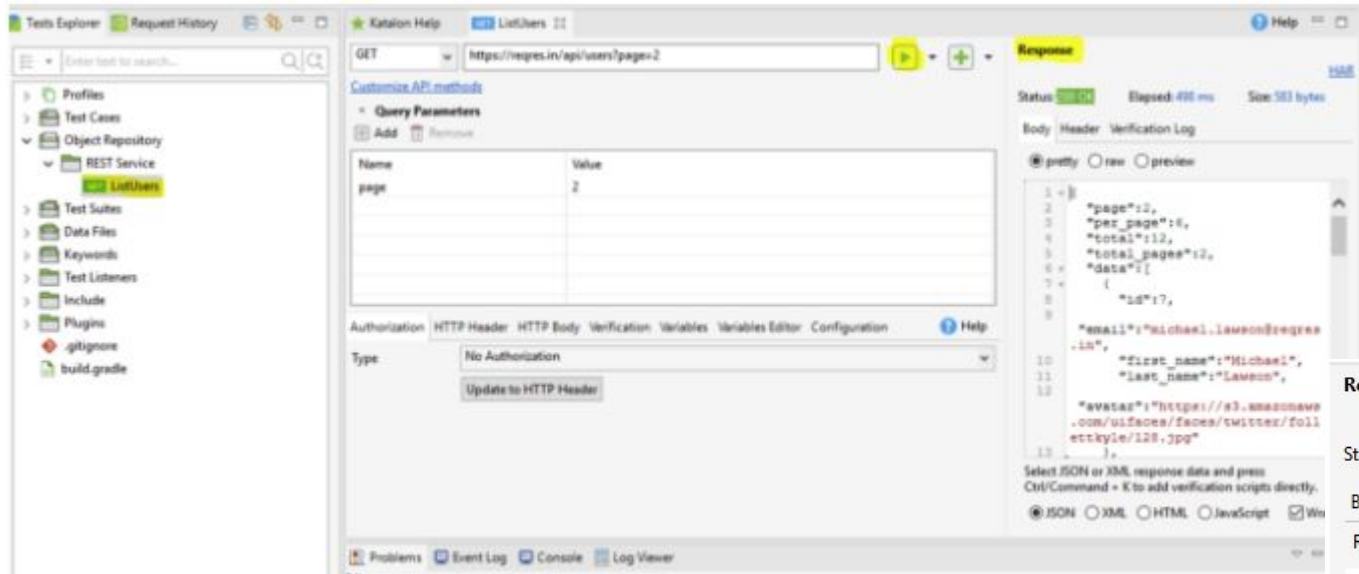
The image shows a 'New' dialog box titled 'Web Service Request'. It contains the following fields and options:

- Name:** ListUsers
- Request Type:** RESTful (selected from a dropdown menu)
- URL:** https://reqres.in/api/users?page=2
- Description:** An empty text area with a vertical scrollbar.

At the bottom right, there are two buttons: 'OK' and 'Cancel'.

REST API

- Setelah itu, klik OK dan akan terbentuk object untuk ListUsers.
- Untuk mengirim request, klik tombol test request (tombol play).
- Tunggu beberapa saat dan response akan segera terlihat



Response

Status: **200 OK** Elapsed: 498 ms Size: 583 bytes

Body Header Verification Log

Result:

Verification log masih kosong

REST API

Mengapa verification log masih kosong? Berarti tidak ada validasi dong?

Iya, karena dia hanya nembak request dan terima respon.

Selain itu tadi kita juga gak menjalankan **test request and verify** karena belum ada langkah verifikasi yang dilakukan. Solusinya ?

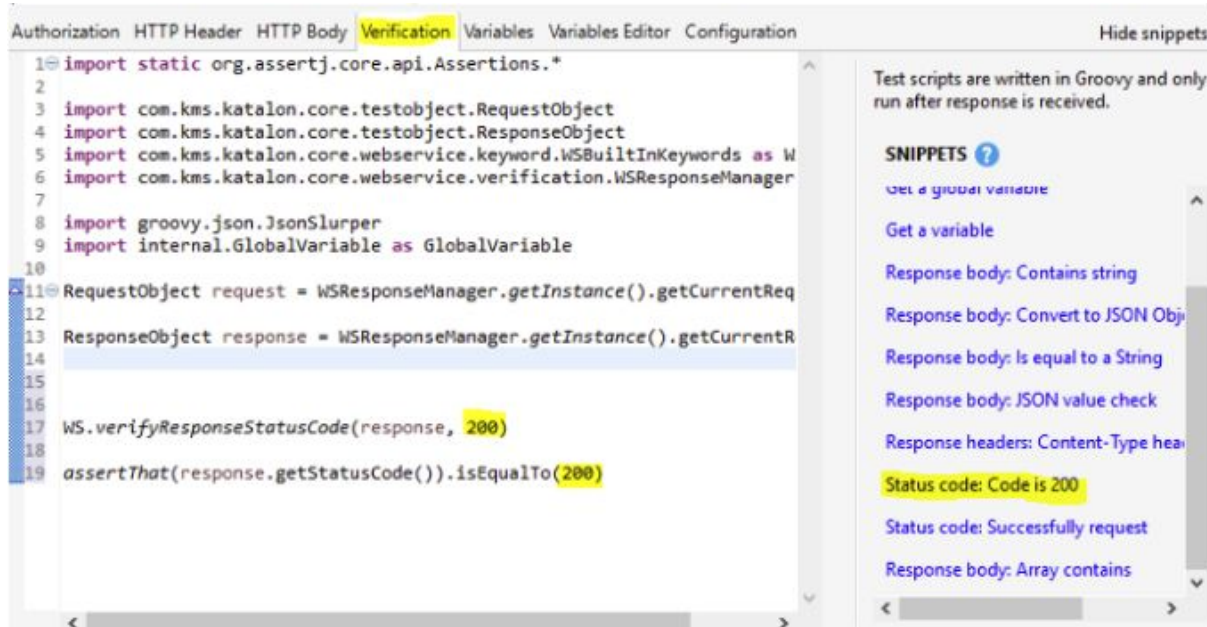
memberikan **verification** agar respon yang didapatkan dicek status atau isinya.

Pertama, kita pastikan respon HTTP yang diterima adalah **HTTP 200** yang berarti sukses.

Caranya ??

REST API

a. klik bar **verification**. Pilih snippet: **“Status code: Code is 200”**. Seharusnya ada code baru yang digenerate oleh Katalon secara otomatis. Status code ini juga bisa diganti tergantung kebutuhan user ya.



The screenshot shows the Katalon Studio interface with the 'Verification' tab selected. The main editor contains Groovy code for verifying the response status code. On the right, a 'SNIPPETS' panel lists various verification snippets, with 'Status code: Code is 200' highlighted.

```
1 import static org.assertj.core.api.Assertions.*
2
3 import com.kms.katalon.core.testobject.RequestObject
4 import com.kms.katalon.core.testobject.ResponseObject
5 import com.kms.katalon.core.webservice.keyword.WSBuiltInKeywords as W
6 import com.kms.katalon.core.webservice.verify.WSResponseManager
7
8 import groovy.json.JsonSlurper
9 import internal.GlobalVariable as GlobalVariable
10
11 RequestObject request = WSResponseManager.getInstance().getCurrentReq
12
13 ResponseObject response = WSResponseManager.getInstance().getCurrentR
14
15
16
17 WS.verifyResponseStatusCode(response, 200)
18
19 assertThat(response.getStatusCode()).isEqualTo(200)
```

Test scripts are written in Groovy and only run after response is received.

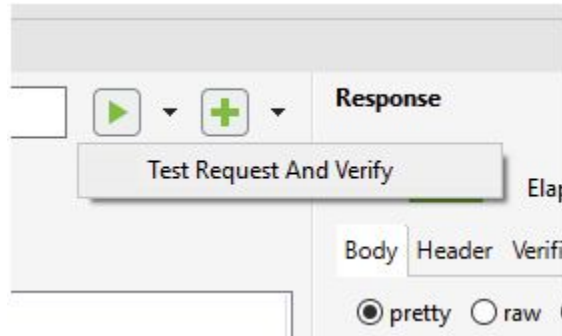
SNIPPETS

- Get a global variable
- Get a variable
- Response body: Contains string
- Response body: Convert to JSON Object
- Response body: Is equal to a String
- Response body: JSON value check
- Response headers: Content-Type header
- Status code: Code is 200**
- Status code: Successfully request
- Response body: Array contains

status code verification di Katalon

REST API

b. Karena kali ini kita menambahkan verification, langkahnya beda. Klik ikon panah kebawah disebelah test request. Kemudian buka verification log lagi.



test request and verify untuk menembak request dan melakukan verifikasi dari script yang dibuat

REST API



The screenshot displays the 'Response' tab of the REST API client in Katalon Studio. The status is '200 OK', the elapsed time is '216 ms', and the size is '583 bytes'. The 'Body' tab is selected, showing a 'Result: PASSED'. Below this, the 'Verification Log' tab is active, displaying a series of log entries and verification steps.

```
2020-03-23 11:25:44.031 INFO c.k.k.core.main.WSVerificationExecutor - -----
2020-03-23 11:25:44.034 INFO c.k.k.core.main.WSVerificationExecutor - START Verification
2020-03-23 11:25:44.699 DEBUG testcase. - 1: request = getInstance().getCurrentRequest()
2020-03-23 11:25:44.897 DEBUG testcase. - 2: response = getInstance().getCurrentResponse()
2020-03-23 11:25:44.948 DEBUG testcase. - 3: verifyResponseStatusCode(response, 200)
2020-03-23 11:25:45.154 DEBUG testcase. - 4: getStatusCode().isEqualTo(200)
2020-03-23 11:25:45.303 INFO c.k.k.core.main.WSVerificationExecutor - END Verification
```

Katalon melakukan verifikasi dari snippet HTTP 200 yang sudah kita tambahkan.

Hasil dari verifikasi diatas adalah PASSED yang artinya Katalon sudah menerima respon dan melakukan verifikasi apakah HTTP response nya 200.

Selanjutnya kita akan melakukan verifikasi mengenai isi dari response JSON yang diterima.

REST API

```
{
  "page": 2,
  "per_page": 6,
  "total": 12,
  "total_pages": 2,
  "data": [
    {
      "id": 7,
      "email": "michael.lawson@reqres.in",
      "first_name": "Michael",
      "last_name": "Lawson",
      "avatar":
        "https://s3.amazonaws.com/uifaces/faces/twitter/follettkyle/128.jpg"
    },
    {
      "id": 8,
      "email": "lindsay.ferguson@reqres.in",
      "first_name": "Lindsay",
      "last_name": "Ferguson",
      "avatar":
        "https://s3.amazonaws.com/uifaces/faces/twitter/araa3185/128.jpg"
    },
    {
      "id": 9,
      "email": "tobias.funke@reqres.in",
      "first_name": "Tobias",
      "last_name": "Funke",
      "avatar":
        "https://s3.amazonaws.com/uifaces/faces/twitter/vivekprvr/128.jpg"
    },
    {
      "id": 10,
      "email": "byron.fields@reqres.in",
      "first_name": "Byron",
      "last_name": "Fields",
      "avatar":
        "https://s3.amazonaws.com/uifaces/faces/twitter/russoedu/128.jpg"
    },
    {
      "id": 11,
      "email": "george.edwards@reqres.in",
      "first_name": "George",
      "last_name": "Edwards",
      "avatar":
        "https://s3.amazonaws.com/uifaces/faces/twitter/mrmoiree/128.jpg"
    },
    {
      "id": 12,
      "email": "rachel.howell@reqres.in",
      "first_name": "Rachel",
      "last_name": "Howell",
      "avatar":
        "https://s3.amazonaws.com/uifaces/faces/twitter/hebertialmeida/128.jpg"
    }
  ],
  "ad": {
    "company": "StatusCode Weekly",
    "url": "http://statuscode.org/",
    "text": "A weekly newsletter focusing on software development,
infrastructure, the server, performance, and the stack end of
things."
  }
}
```

REST API

Sama seperti langkah sebelumnya, namun kali ini kita pakai script untuk **memvalidasi value dari JSON**.

Katalon sudah menyediakan snippet untuk membantu kita supaya tidak perlu menulis script dari awal. Pilih snippet: ***“Response Body: JSON value check”***.

```
WS.verifyElementPropertyValue(response,  
'issues[0].fields.project.key', 'KTP')
```

Bagaimana cari path JSON nya?

- response: script ini mengambil value response yang sudah dideskripsikan sebelumnya.
- 'issues[0].fields.project.key': path yang akan divalidasi
- 'KTP': value apa yang seharusnya ada dalam path tersebut

REST API

CTRL + K (untuk windows).

Pertama, klik dulu pada bagian apa di response yang ingin di validasi.

Klik pada tulisan “Lindsay”.



REST API

Lakukan **CTRL + K** lagi
seharusnya akan ada script yang digenerate di bagian verification, seperti berikut:

```
WS.verifyElementPropertyValue(response, 'data[1].first_name',  
"Lindsay")
```

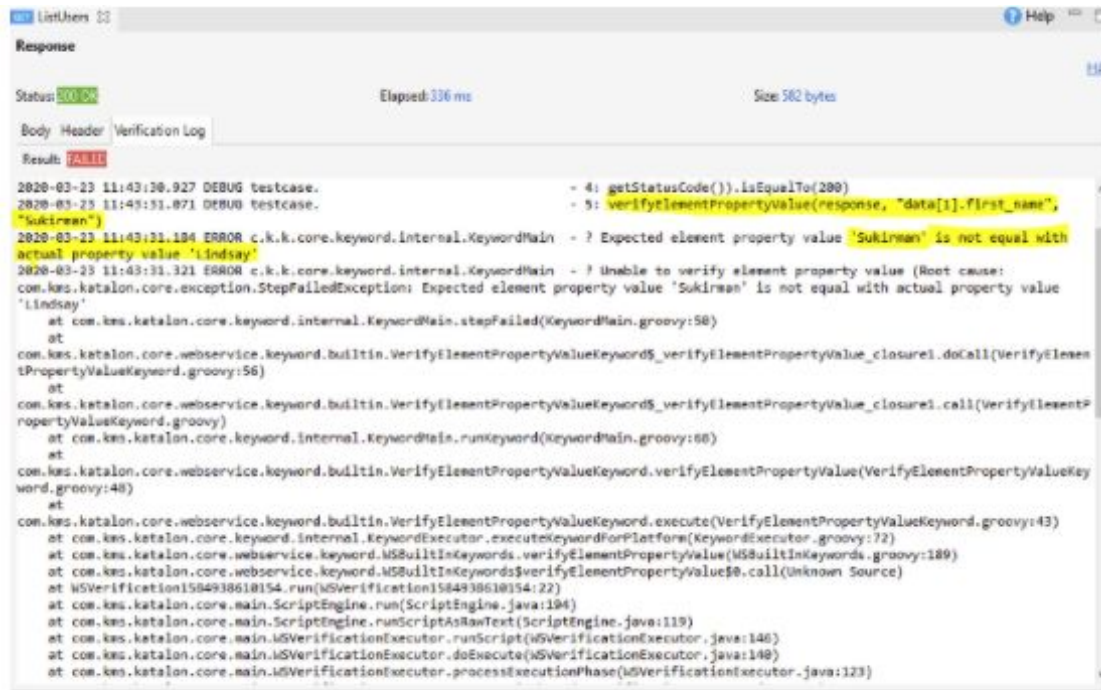
Mari kita cek di **verification lognya**:



Nah. Hasil dari verification tersebut adalah **PASSED**. Berarti sudah sama dengan apa yang kita harapkan.

CONCLUSION

Memang gimana kalo bukan PASSED? Oke. Kita kembali ke verification dan ganti Lindsay menjadi Amel. Maka setelah di test request and verify, hasilnya adalah:



The screenshot shows the Karbon test runner interface. At the top, it says 'Response' and 'Status: 200 OK'. Below that, it says 'Elapsed: 336 ms' and 'Size: 562 bytes'. The 'Body' tab is selected, showing a 'Verification Log'. The log contains several lines of text, including timestamps and error messages. The most relevant part is the error message: 'Expected element property value 'Sukirman' is not equal with actual property value 'Lindsay''. This indicates that the test failed because the actual value 'Lindsay' did not match the expected value 'Sukirman'.

```
2020-03-23 11:43:30.927 DEBUG testcase. - 4: getStatusCode().isEqualTo(200)
2020-03-23 11:43:31.071 DEBUG testcase. - 5: verifyElementPropertyValue(response, "data[1].first_name",
"Sukirman")
2020-03-23 11:43:31.154 ERROR c.k.k.core.keyword.internal.KeywordMain - ? Expected element property value 'Sukirman' is not equal with
actual property value 'Lindsay'
2020-03-23 11:43:31.321 ERROR c.k.k.core.keyword.internal.KeywordMain - ? Unable to verify element property value (Root cause:
com.kms.katalon.core.exception.StepFailedException: Expected element property value 'Sukirman' is not equal with actual property value
'Lindsay'
    at com.kms.katalon.core.keyword.internal.KeywordMain.stepFailed(KeywordMain.groovy:58)
    at
    com.kms.katalon.core.webservice.keyword.builtin.VerifyElementPropertyValueKeyword.verifyElementPropertyValue_closure1.doCall(VerifyElemen
tPropertyValueKeyword.groovy:56)
    at
    com.kms.katalon.core.webservice.keyword.builtin.VerifyElementPropertyValueKeyword.verifyElementPropertyValue_closure1.call(VerifyElementP
ropertyValueKeyword.groovy)
    at com.kms.katalon.core.keyword.internal.KeywordMain.runKeyword(KeywordMain.groovy:66)
    at
    com.kms.katalon.core.webservice.keyword.builtin.VerifyElementPropertyValueKeyword.verifyElementPropertyValue(VerifyElementPropertyValueKey
word.groovy:48)
    at
    com.kms.katalon.core.webservice.keyword.builtin.VerifyElementPropertyValueKeyword.execute(VerifyElementPropertyValueKeyword.groovy:43)
    at com.kms.katalon.core.keyword.internal.KeywordExecutor.executeKeywordForPlatform(KeywordExecutor.groovy:72)
    at com.kms.katalon.core.webservice.keyword.WSBuiltInKeywords.verifyElementPropertyValue(WSBuiltInKeywords.groovy:189)
    at com.kms.katalon.core.webservice.keyword.WSBuiltInKeywords.verifyElementPropertyValue(WSBuiltInKeywords.groovy:189)
    at WSVerification1584938618154.run(WSVerification1584938618154:22)
    at com.kms.katalon.core.main.ScriptEngine.run(ScriptEngine.java:194)
    at com.kms.katalon.core.main.ScriptEngine.runScriptAsRawText(ScriptEngine.java:119)
    at com.kms.katalon.core.main.WSVerificationExecutor.runScript(WSVerificationExecutor.java:146)
    at com.kms.katalon.core.main.WSVerificationExecutor.doExecute(WSVerificationExecutor.java:148)
    at com.kms.katalon.core.main.WSVerificationExecutor.processExecutionPhase(WSVerificationExecutor.java:123)
```

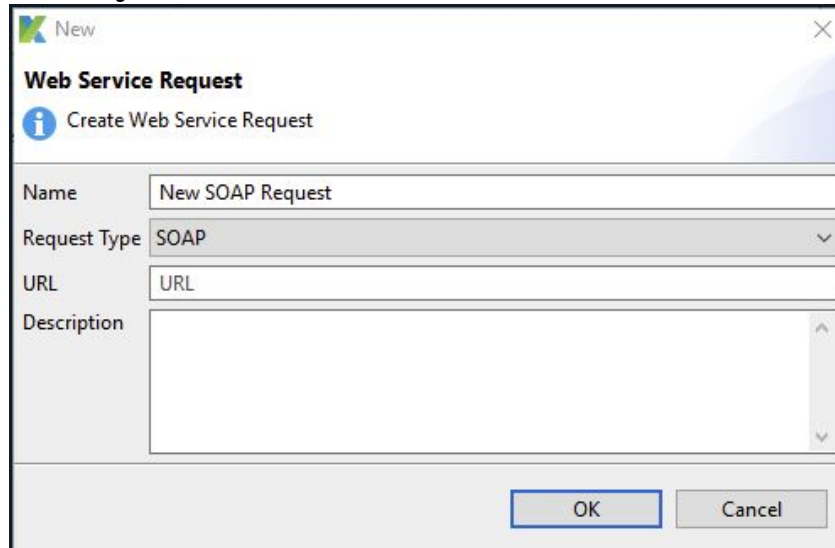
Hasilnya adalah **FAILED** karena valuenya tidak sama seperti yang diharapkan

SOAP API

When sending a SOAP Request in Katalon Studio, you can receive a response from the API server for examination and troubleshooting.

Creating a SOAP-based Request

1. From the main menu, select File > New > Web Service Request.
2. In the New Web Service Request dialog, select SOAP in the Request Type list and click OK to create a new SOAP object.



A New request object is created under the Object Repository of Katalon Studio

SOAP API

Adding SOAP Request Details

After you've created a request successfully, double-click on the request to open its editor for adding details. In the opened editor of the New Request object, you can see all the required information of a request object.

The screenshot shows the 'New Request' editor in Katalon Studio. The title bar includes 'Katalon Help', a 'SOAP' tab, and 'New Request (4)'. The main area has a 'SOAP' dropdown menu, a 'WSDL URL' text field, a 'Service Function' dropdown menu, and a 'Service Endpoint' text field. To the right of these fields are buttons for 'Load Service Function' and 'Load New Content'. At the bottom, there is a tabbed interface with 'Authorization' selected, showing a 'Type' dropdown menu set to 'No Authorization' and an 'Update to HTTP Header' button. Other tabs include 'HTTP Header', 'Request Message', 'Verification', 'Variables', 'Variables Editor', and 'Configuration'. A 'Help' button is also present.

★ Katalon Help SOAP New Request (4) ⌵

SOAP ▼

▶ + ▼

WSDL URL Load Service Function

Service Function ▼ Load New Content

Service Endpoint

Authorization HTTP Header Request Message Verification Variables Variables Editor Configuration ? Help

Type ▼

Update to HTTP Header

SOAP API

Request Method

The request method indicates the expected action to be executed on the specified resource. Katalon Studio supports the following SOAP methods: SOAP, SOAP 1.2, POST, GET. By default, Katalon selects SOAP as a method for a new SOAP request.

The screenshot shows the Katalon Studio SOAP request configuration window. At the top, there is a 'Katalon Help' link and a 'SOAP Add' button. Below this, a dropdown menu is open, showing 'SOAP' as the selected method, with other options being 'SOAP12', 'GET', and 'POST'. To the right of the dropdown, there are two buttons: a green play button and a green plus button. Below the dropdown, there is a text input field containing the URL 'http://www.dneonline.com/calculator.asmx?wsdl'. To the right of this field is a 'Load Service Function' button. Below the URL field, there is an 'Add' button and a dropdown menu. To the right of the 'Add' button is a 'Load New Content' button. Below these buttons, there is a checkbox labeled 'Use the endpoint and SOAPAction header parsed from WSDL' which is checked. To the right of the checkbox is a blue question mark icon. Below the checkbox, there is a 'Service Endpoint' label and an empty text input field. At the bottom of the window, there is a tabbed interface with tabs for 'Authorization', 'HTTP Header', 'Request Message', 'Verification', 'Variables', 'Variables Editor', and 'Configuration'. The 'Authorization' tab is currently selected. Below the tabs, there is a 'Type' label and a dropdown menu showing 'No Authorization'. To the right of the dropdown menu is a blue question mark icon and a 'Help' link. Below the dropdown menu, there is an 'Update to HTTP Header' button.

SOAP API

WSDL URL

This field is for a WSDL path from which Katalon Studio imports the content to this SOAP request.

The screenshot shows the 'New Request' configuration window in Katalon Studio. The 'SOAP' tab is selected. The 'WSDL URL' field is highlighted with a blue border and contains the text 'http://www.dneonline.com/calculator.asmx?WSDL'. To the right of the 'WSDL URL' field are two buttons: 'Load Service Function' and 'Load New Content'. Below the 'WSDL URL' field are fields for 'Service Function' and 'Service Endpoint'. At the bottom, there is a tabbed interface with 'Authorization' selected, showing 'Type' as 'No Authorization' and an 'Update to HTTP Header' button. Other tabs include 'HTTP Header', 'Request Message', 'Verification', 'Variables', 'Variables Editor', and 'Configuration'. A 'Help' button is located in the top right corner of the bottom section.

Katalon Help SOAP New Request (4) SOAP Add

SOAP ▼

▶ + ▼

WSDL URL Load Service Function

Service Function ▼ Load New Content

Service Endpoint

Authorization HTTP Header Request Message Verification Variables Variables Editor Configuration ? Help

Type ▼

Update to HTTP Header

SOAP API

Service Function

The function that you want to use in this SOAP request. When clicking Load Service Function, you can retrieve a list of service functions available from the WSDL file.

The screenshot shows the Katalon Test Runner interface for configuring a SOAP API request. The top bar includes a 'Katalon Help' link, a tab for '*New Request (4)', and an 'Add' button. Below the top bar, there is a 'SOAP' dropdown menu and a 'WSDL URL' field containing 'http://www.dneonline.com/calculator.asmx?WSDL'. To the right of the WSDL URL field are two buttons: 'Load Service Function' and 'Load New Content'. Below the WSDL URL field is a 'Service Function' dropdown menu with a list of functions: 'Add', 'Subtract', 'Multiply', and 'Divide'. The 'Add' function is currently selected. To the right of the Service Function dropdown is a 'Service Endpoint' field. Below the Service Function dropdown is an 'Authorization' dropdown menu with 'No Authorization' selected. To the right of the Authorization dropdown is a 'Type' dropdown menu with 'No Authorization' selected. At the bottom right, there is a 'Help' button.

Katalon Help *New Request (4) SOAP Add

SOAP

WSDL URL Load Service Function

Service Function Add Load New Content

Service Endpoint Add Subtract Multiply Divide

Authorization HTTP No Authorization ? Help

Type No Authorization Update to HTTP Header

SOAP API

Each Service Function carries its own content, including Service Endpoint, SOAPAction Header and Request message.

In Service Endpoint, You can specify another URL indicating the desired service endpoint of this request.

The screenshot shows the 'SOAP' configuration tab in Katalon Help. The interface includes a top bar with 'Katalon Help', 'New Request (4)', and 'Add' buttons. Below this, there's a 'SOAP' dropdown menu and two icons (a play button and a plus button). The main configuration area contains three input fields: 'WSDL URL' with the value 'http://www.dneonline.com/calculator.asmx?WSDL', 'Service Function' with the value 'Add', and 'Service Endpoint' with the value 'http://www.dneonline.com/calculator.asmx'. To the right of these fields are two buttons: 'Load Service Function' and 'Load New Content'. Below the input fields is a tabbed interface with 'Authorization', 'HTTP Header', 'Request Message', 'Verification', 'Variables', 'Variables Editor', and 'Configuration'. The 'HTTP Header' tab is currently selected. At the bottom, there's a table with two columns: 'Name' and 'Value'. The table contains two rows: 'SOAPAction' with the value 'http://tempuri.org/Add' and 'Content-Type' with the value 'text/xml; charset=utf-8'.

| Name | Value |
|--------------|-------------------------|
| SOAPAction | http://tempuri.org/Add |
| Content-Type | text/xml; charset=utf-8 |

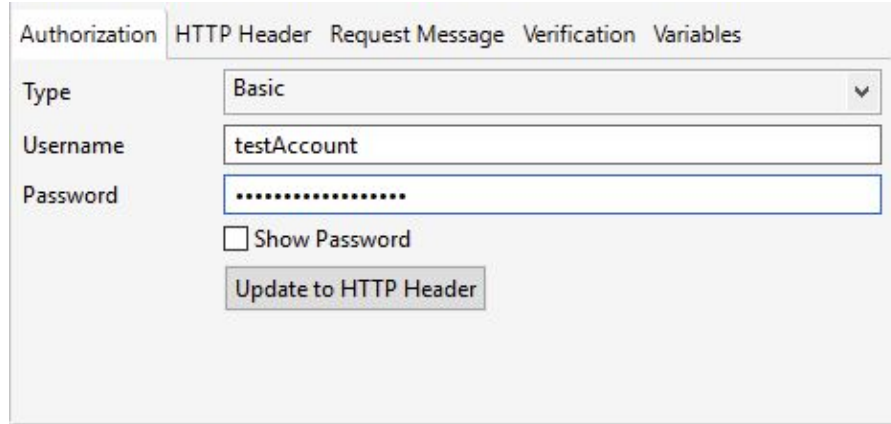
SOAP API

Request Authentication

This part is used for authenticating and authorizing the request, which means to verify if the client is permitted to send the request and to perform the endpoint operation.

For more details on using each type of auth, please see:

- [Basic](#)
- [OAuth 1.0](#)
- [OAuth 2.0](#)



The screenshot shows a configuration window with five tabs: Authorization, HTTP Header, Request Message, Verification, and Variables. The 'Authorization' tab is selected. It contains a 'Type' dropdown menu set to 'Basic', a 'Username' text field with 'testAccount', and a 'Password' text field with masked characters. Below the password field is an unchecked checkbox labeled 'Show Password' and a button labeled 'Update to HTTP Header'.

| Authorization | HTTP Header | Request Message | Verification | Variables |
|--|-------------|-----------------|--------------|-----------|
| Type | Basic | | | |
| Username | testAccount | | | |
| Password | | | | |
| <input type="checkbox"/> Show Password | | | | |
| <button>Update to HTTP Header</button> | | | | |

SOAP API

Request Headers

The header information needs sending along with this SOAP request. You can select headers from the list of suggested options (by double-clicking on the Name cell) or enter another header of your interest.

The screenshot shows the Katalon Studio SOAP client interface. At the top, there's a tab bar with 'Katalon Help', 'SOAP *New Request (4)', and 'SOAP Add'. Below this is a 'SOAP' dropdown menu and two buttons: a green play button and a green plus button. The main configuration area includes fields for 'WSDL URL' (http://www.dneonline.com/calculator.asmx?WSDL), 'Service Function' (Add), and 'Service Endpoint' (http://www.dneonline.com/calculator.asmx). To the right of these fields are buttons for 'Load Service Function' and 'Load New Content'. Below the configuration area is a tab bar with 'Authorization', 'HTTP Header', 'Request Message', 'Verification', 'Variables', 'Variables Editor', and 'Configuration'. The 'HTTP Header' tab is selected, showing a table with 'Name' and 'Value' columns. The table contains two rows: 'SOAPAction' with value 'http://tempuri.org/Add' and 'Content-Type' with value 'text/xml; charset=utf-8'. Above the table are '+ Add' and 'Remove' buttons.

| Name | Value |
|--------------|-------------------------|
| SOAPAction | http://tempuri.org/Add |
| Content-Type | text/xml; charset=utf-8 |
| | |
| | |

SOAP API

Request Message

The information that you want to transmit in this SOAP request. You can get it after clicking Load New Content of the selected service function.

The screenshot displays the Katalon Help SOAP configuration window. The top bar includes a star icon, 'Katalon Help', a tab for '*New Request (4)', and an 'Add' button. Below this, a 'SOAP' dropdown menu is on the left, and a play button and a plus button are on the right. The main configuration area contains three fields: 'WSDL URL' with the value 'http://www.dneonline.com/calculator.asmx?WSDL', 'Service Function' with the value 'Add', and 'Service Endpoint' with the value 'http://www.dneonline.com/calculator.asmx'. To the right of these fields are two buttons: 'Load Service Function' and 'Load New Content'. Below the configuration fields is a tabbed interface with the following tabs: 'Authorization', 'HTTP Header', 'Request Message' (which is the active tab), 'Verification', 'Variables', 'Variables Editor', and 'Configuration'. The 'Request Message' tab shows a SOAP XML request structure with line numbers 1 through 9 on the left. The XML content is as follows:

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2   xmlns:tem="http://tempuri.org/"
3   <soapenv:Header/>
4   <soapenv:Body>
5     <tem:Add>
6       <tem:intA>3</tem:intA>
7       <tem:intB>3</tem:intB>
8     </tem:Add>
9   </soapenv:Body>
</soapenv:Envelope>
```


SOAP API

After sending the service request, Katalon Studio retrieves a message from the server and displays it in the Response view of the request. A service response comprises Status, Elapsed time, and Size fields; Body section, Header, and Verification Log.

- Status: The status code of the response
- Elapsed: The total time that starts from the request is sent until Katalon Studio receives the last byte of the response
- Size: Size of the response package

SOAP API

Response Body

There are 2 viewing formats: pretty and raw. For example, the SOAP's response to <http://www.dneonline.com/calculator.asmx?WSDL> is shown below.

Response [HAR](#)

Status: **200 OK** Elapsed: 664 ms Size: 184 bytes

Body Header Verification Log

☒ pretty ☐ raw

```
1 <?xml version="1.0" encoding="utf-
2 <soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.o
  rg/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XM
  LSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XM
  LSchema">
3   <soap:Body>
4     <AddResponse
  xmlns="http://tempuri.org/">
5       <AddResult>6
6       </AddResult>
7     </AddResponse>
8   </soap:Body>
9 </soap:Envelope>
10
```

SOAP API

Response

[HAR](#)Status: **200 OK**

Elapsed: 664 ms

Size: 184 bytes

Body

Header

Verification Log



pretty



raw

1

```
<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><AddResponse
xmlns="http://tempuri.org/">
<AddResult>6</AddResult>
</AddResponse></soap:Body>
</soap:Envelope>
```