JSON Response Validation Scenarios

Let us see, how various types of responses can be validated in vREST.

1. My API returns the static response.
2. I just want to validate the schema of my response, not the actual content.
3. My API returns some dynamic properties like _id, createdOn etc. and I want to ignore them during response validation.
4. My API returns a very large response and I am interested in validating only a small part of my API response.
5. My API returns some response in which some part of the response can be obtained from the responses of previous test cases.
6. My API returns dynamic response and none of the above fit to my needs.

Scenario 1: My API returns the static response.

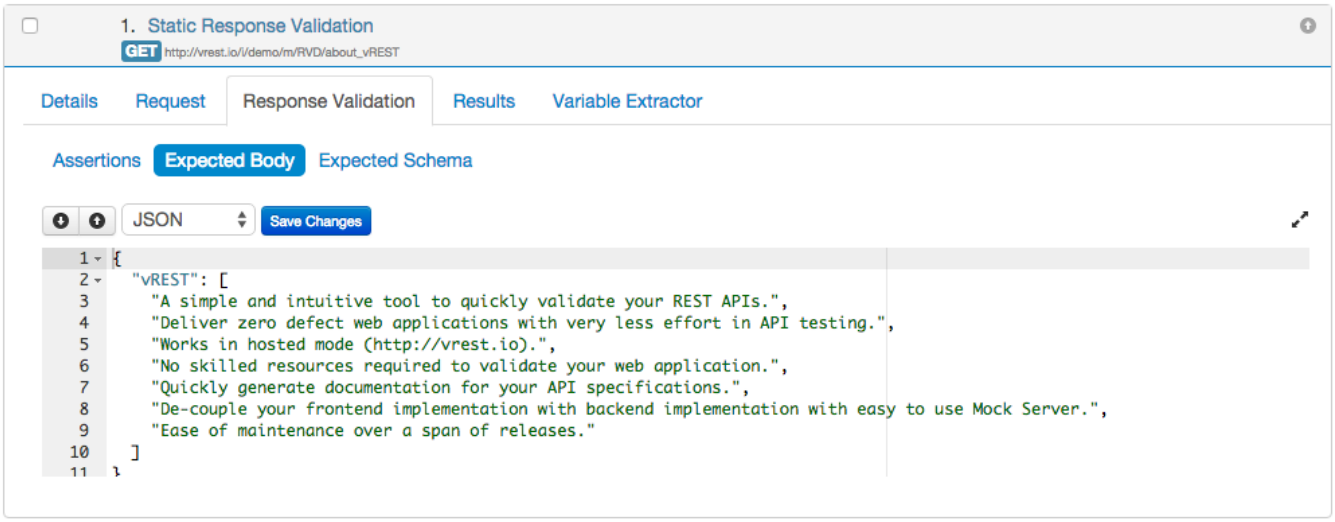
If your API returns the static response, then just set the Expected Body with the static response and Expected Status Code.

Let us suppose, your API returns the following static JSON response,

Status Code: 200

```
{
  "vREST": [
    "A simple and intuitive tool to quickly validate your REST APIs.",
    "Deliver zero defect web applications with very less effort in API testing.",
    "Works in hosted mode (http://vrest.io).",
    "No skilled resources required to validate your web application.",
    "Quickly generate documentation for your API specifications.",
    "De-couple your frontend implementation with backend implementation with easy to use Mock Server.",
    "Ease of maintenance over a span of releases."
  ]
}
```

For such scenarios, simply specify your static response body in Expected Body and set the Expected Status Code also as specified in the following image. Validator used for this scenario is "Default Validator".



Scenario 2: I just want to validate the schema of my response, not the actual content.

Let us consider a scenario, where response contains random data (unpredictable data) or list of some unordered items. Only the schema/structure of the response is predictable. In such scenario, Default Schema Validator can be used.

e.g. let's suppose an API returns a list of random quotes. Since it returns a random list, so we cannot predict the outcome of this API. But what we know is the structure/schema of this response.

In the following response,

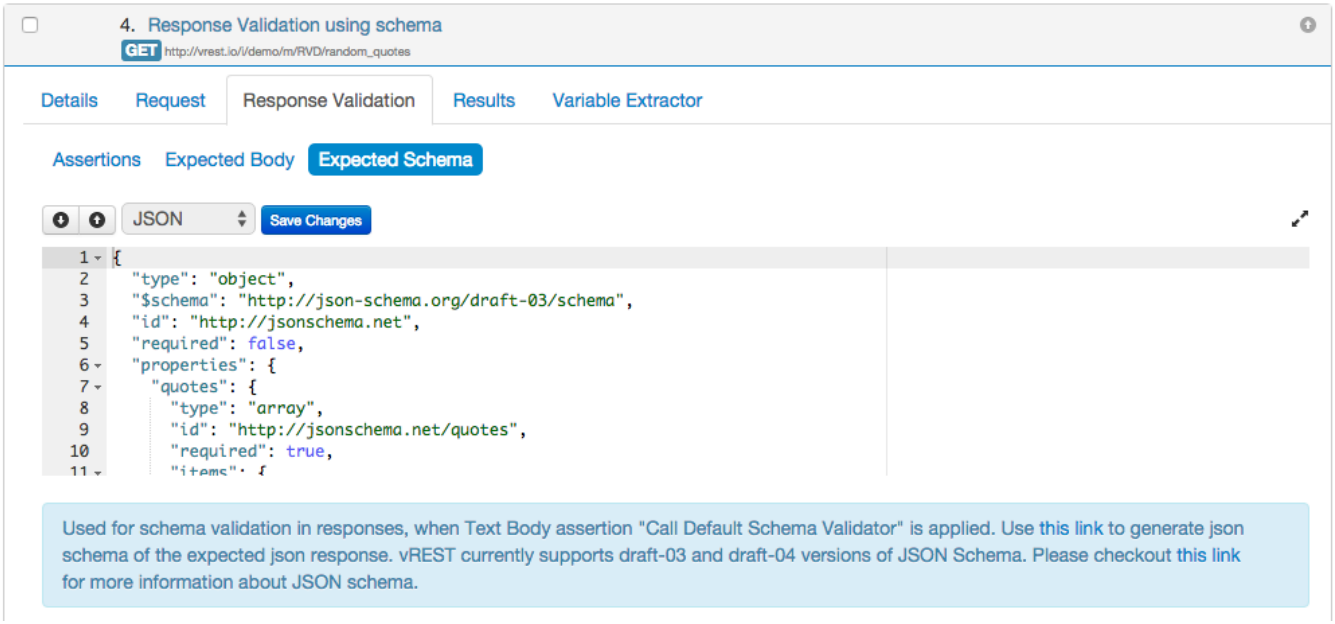
1. Every item in the quotes list have fields like "quote", "author", "designation", "organization".

2. And each field is of type String.

Status Code: 200


```
{
  "quotes": [
    {
      "quote": "Stay hungry, stay foolish.",
      "author": "Steve Jobs",
      "designation": "Co-founder, Chairman & CEO",
      "organization": "Apple Inc."
    },
    {
      "quote": "If you're changing the world, you're working on important things. You're excited to get up in
the morning.",
      "author": "Larry Page",
      "designation": "Co-founder & CEO",
      "organization": "Google"
    },
    ...
  ]
}
```


For using Default Schema Validator, first we need to define the Expected Schema of the response as shown in the following image. We can use an external tool http://jsonschema.net to automatically generate the schema of our response.



Note:

- If you use Default Schema Validator, you need to define the expected schema of the response.
- We do not own the utility hosted at http://jsonschema.net. The link provided here is just for convenience in writing JSON schema.
- The above utility provides you the basic JSON schema of your response plus some constraints. If you want to provide additional schema constraints, I recommend you to read about JSON Schema.

Scenario 3: My API returns some dynamic properties like _id, createdOn etc. and I want to ignore them during response validation and also I want to validate some of the properties with regular expression.

Suppose, If your API which creates a resource on the server and returns some dynamic properties like _id, createdOn etc. And you want to ignore these dynamic properties during the response validation and also you want to check the email property with email regular expression. In such scenario, Default Validator can be used.

e.g. Let us suppose, the API returns the following response:

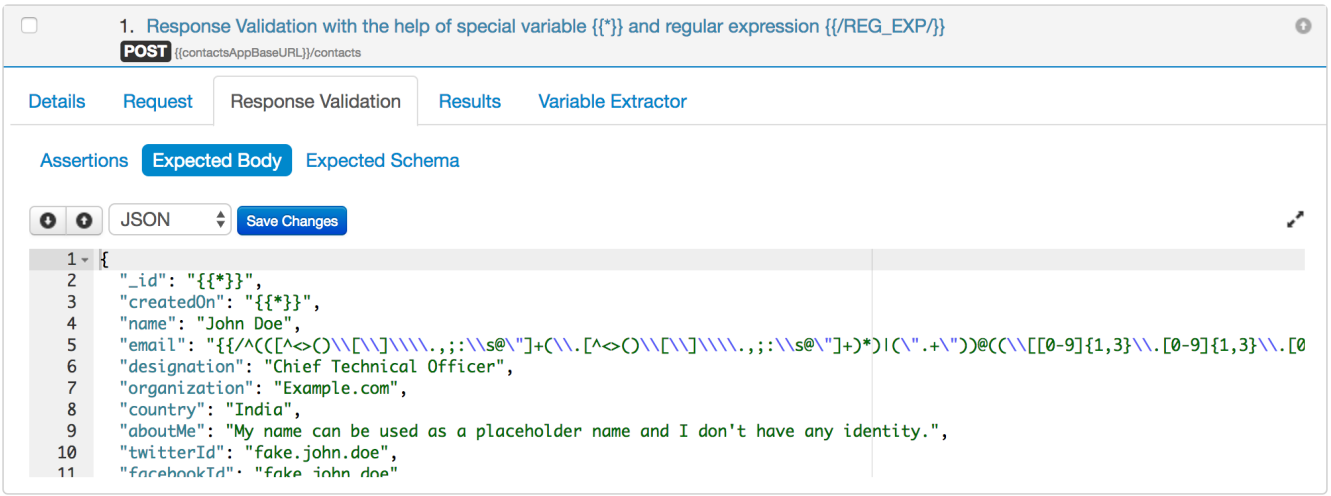
Status Code: 200


```
{
  "_id": "536493015f56452a03000010",
  "createdOn": "2014-05-03T06:56:01.134Z",
  "name": "John Doe",
  "email": "john.doe@example.com",
  "designation": "Chief Technical Officer",
  "organization": "Example.com",
  "country": "India",
  "aboutMe": "My name can be used as a placeholder name and I don't have any identity.",
  "twitterId": "fake.john.doe",
  "facebookId": "fake.john.doe",
  "githubId": "fake.john.doe"
}
```


Here in the above response, you want to ignore the dynamically generated _id and createdOn field.
For such scenarios,

1. Simply use the special variable "{{*}}" for values, which you want to ignore.
2. And use regular expression in format "{{**/REG_EXP/**}}" to match the value against a regular expression.

Now, the expected body should look like this:



Note:

- Special variable must always be enclosed with double quotes.
- Javascript regular expression must be used and must be escaped to be used as string. For escaping the regular expression, you may also use any third party tools like FreeFormatter.

Scenario 4: My API returns a very large response and I am interested in validating only a small part of my API response.

If you want to validate only a small part of your API response and want to ignore rest of the properties then you can use special variable **"{{*}}": "{{*}}".**

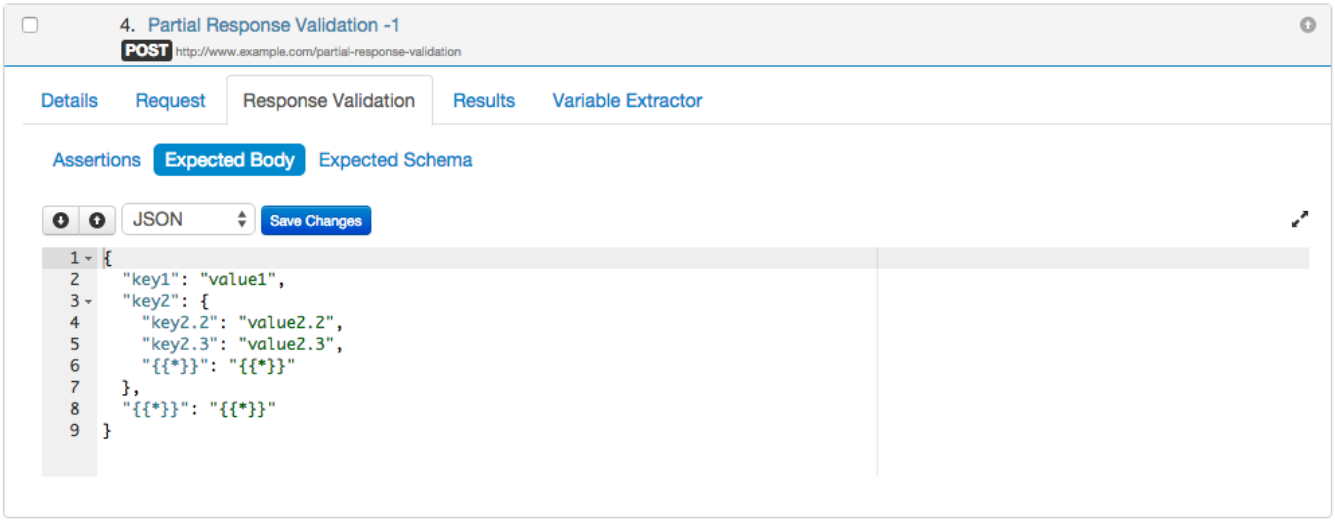
Let us suppose, the API returns the following response:

Status Code: 200

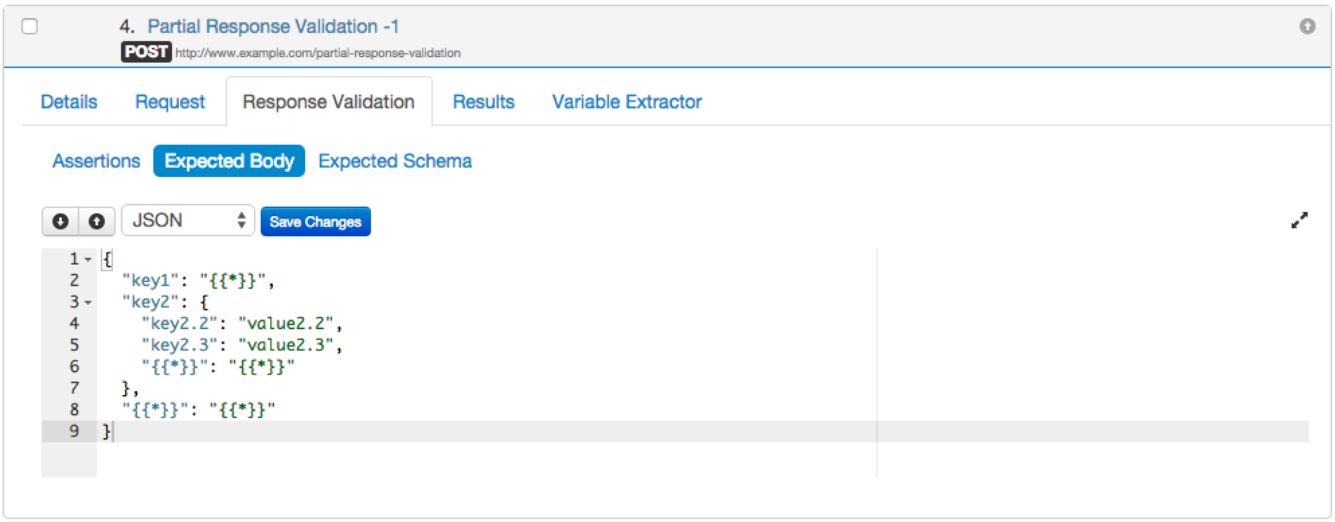
```
{
  "key1": "value1",
  "key2": {
    "key2.1": "value2.1",
    "key2.2": "value2.2",
    "key2.3": "value2.3",
    "key2.4": "value2.4"
  },
  "key3": "value3",
  "key4": "value4"
}
```

And in the above response, you only want to validate key1, key2.2 and key2.3 values.

For such scenarios, simply use the special variable "{{*}}": "{{*}}" (key-value pair) to ignore rest of the keys and values. Now, the expected body should look like this:



Further, let us suppose, you want to validate only the existence of key "key1" in your response, not the value of "key1", then you can mix this scenario with scenario 4 and write your expected body like this:



Scenario 5: My API returns some response in which some part of the response can be obtained from the responses of previous test cases.

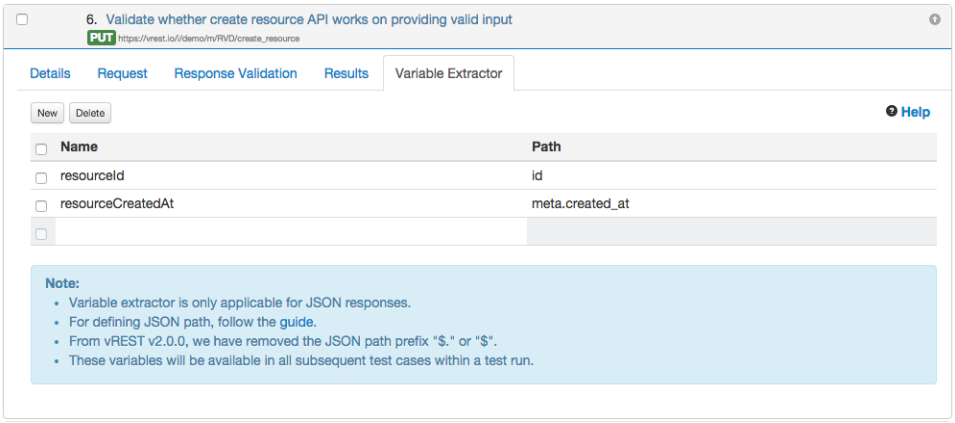
Let us take an example in which one test case creates a resource on server and second test case updates that newly created resource.

- Suppose you have an API which creates resource on server and returns the following JSON response:

Status Code: 200

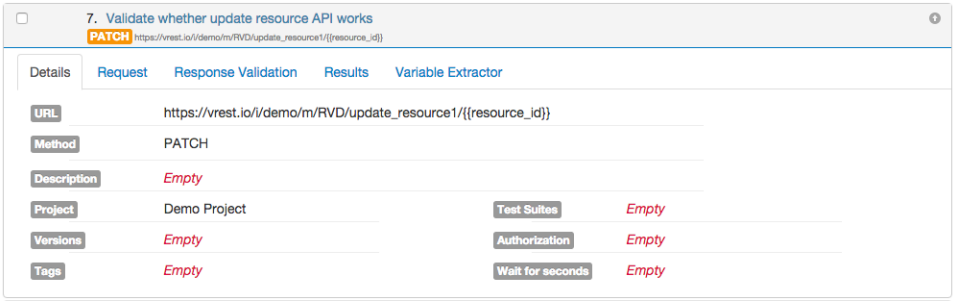
```
{
  "id": "54a79b704cba8d5328d087f5",
  "resource_name": "testcase",
  "resource_url": "http://vrest.io/i/demo/m/RVD/create_resource",
  "resource_description": "This API creates a resource on the server",
  "meta": {
        "created_at": "2015-01-03T07:41:21.000Z"
  }
}
```

- Now, you can save the id of newly created resource into variable say "resourceId" and creation time into variable say "resourceCreatedAt". You can extract these variables in the following way:



Few points regarding writing **Path** in the above table:
  - Each individual property value can be extracted via JSON Path expression e.g. `id` or `meta.created_id`
  - For more information, read JSON Path syntax
- Now you can use these extracted variables in subsequent requests. Note that once a variable is defined, it can be used in all subsequent requests within that test run only. If you want to override this variable, simply re-define the variable in any request.

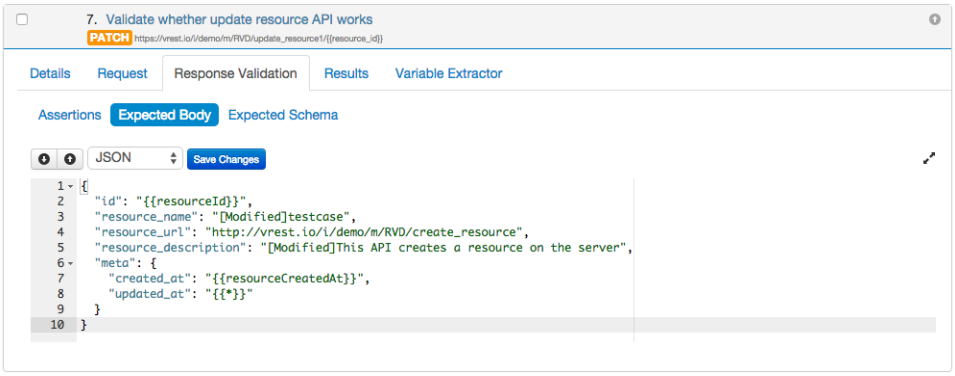
  Now, suppose you have an API which updates this newly created resource and it needs the ID of the resource to update. You can use the {{resourceId}} variable (extracted in previous step) in the URL as shown in the following figure:



- And let us suppose, the Update API returns the following response:

```
{
  "id": "54a79b704cba8d5328d087f5",
  "resource_name": "[Modified]testcase",
  "resource_url": "http://vrest.io/i/demo/m/RVD/create_resource",
  "resource_description": "[Modified]This API creates a resource on the server",
  "meta": {
        "created_at": "2015-01-03T07:41:21.000Z",
        "updated_at": "2015-01-03T07:51:01.000Z"
  }
}
```

Now, you can write our expected body like this:



Note: In the above test case, fields "id" and "created_at" will be replaced from the values extracted from previous test case, and updated_at value will be replaced from the value received the actual body, before response validation. So, we can use Default Validator in such scenarios.

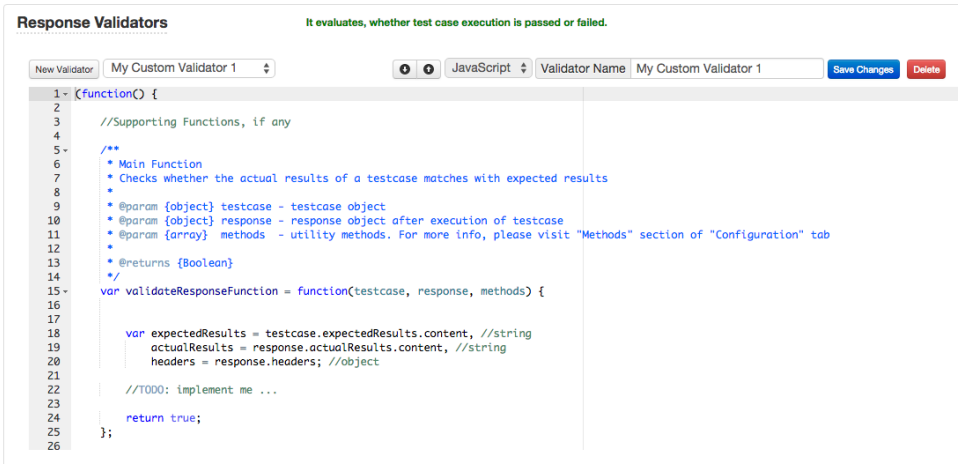
Scenario 6: My API returns dynamic response and none of the above fit to my needs.

In vREST, most of the responses can be validated with the help of built-in response validators "Default Validator" and "Default Schema Validator". But if that doesn't fit your needs, then you can define your own custom validator in vREST or you can even mix and match validators and assertions.

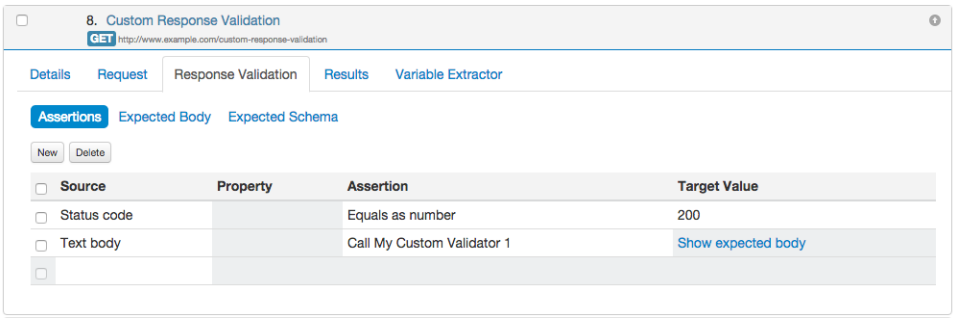
- Custom Validator is basically a Javascript function and vREST invokes this function, if you associate your custom defined validator with the test case.
- vREST provides expected response and actual response to this function.
- Now, its your job to validate expected response with actual response.
- If you return true then vREST will mark the test case as passed and otherwise failed.

For this scenario,

1. First define your custom validator in **Project Configuration** >> **Response Validator Section.**



2. Now associate this custom validator with your test case like below:



That's it. Now your test case will be validated with your custom validator when you execute it.

Note

If you think, your scenario is not covered here then you can discuss your scenario with us by sending an email to "support@vrest.io".

Note:

- The above scenarios are also covered in our online demo available at Response Validation Demo
- Variables plays an important role in response validation. See also Environments / Variables