A Developer Diary
{about:"code learn and share"}

January 24, 2018 By Abhisek Jana — 1 Comment (Edit)

How to Automate REST API JSON Schema Validation testing using Postman

In recent years with the advent of cloud and modern UI technology, we have decentralized and distributed the processing power, storages and applications across many geographies. The middleware which connects a user with the backend is nothing but RESTful Services. Hence lightweight REST Services have become ubiquitous across many industries. In this tutorial we will learn How to Automate REST API JSON Schema Validation testing using Postman.

As we have seen in the spider man movie, power comes with responsibilities. So the testing the REST Services has become very critical part of the development workflow/lifecycle.

Background

Any modern development workflow will have some sort of Software Delivery Automation (SDA), such as Atlassian stack or other open source stack. We need to have a process to execute the automated test cases for the REST Services which can be run along with the build process.

It's very time consuming to test a JSON based REST Service without a valid Schema. JSON was built to be schema less, however recently the community has found the need for having JSON Schema. Hence there are unofficial versions of that we can use (http://json-schema.org/).

Why Schema Validation

Validating the structure of the RESTful services using a schema is simple, otherwise it would be almost impossible to test each fields manually every time there is a change. You can always add additional test scripts as part of your test scripts.

Postman

Postman has become as very popular choice for RESTFul API testing (https://www.getpostman.com/) in recent years. It's mostly free and supports many advanced features and functionalities.

Tools

We will be using following tools in this tutorial. Please install them as needed.

- Postman (https://www.getpostman.com/)
- newman (https://www.npmjs.com/package/newman)
  - Install using `npm install newman –global`
- App Server/Web Server to deploy our API
  - I will be using static JSON for the test, hence I use my local MAMP (webserver). You can use anything of your choice and need,however when you integrate this with the build process you need thses as part of the dev/non-prod server.

High Level Steps:

Here are the high level steps we will be working on:

1. Generate JSON Schema
2. Configure and Test using Postman
3. Run from newman

Generate JSON Schema:

Assume that you already have a JSON structure we will start with the Schema Generation. In order to make our example simple we will use following services.

`http://localhost:8888/APITesting/Service/searchStudents.json`

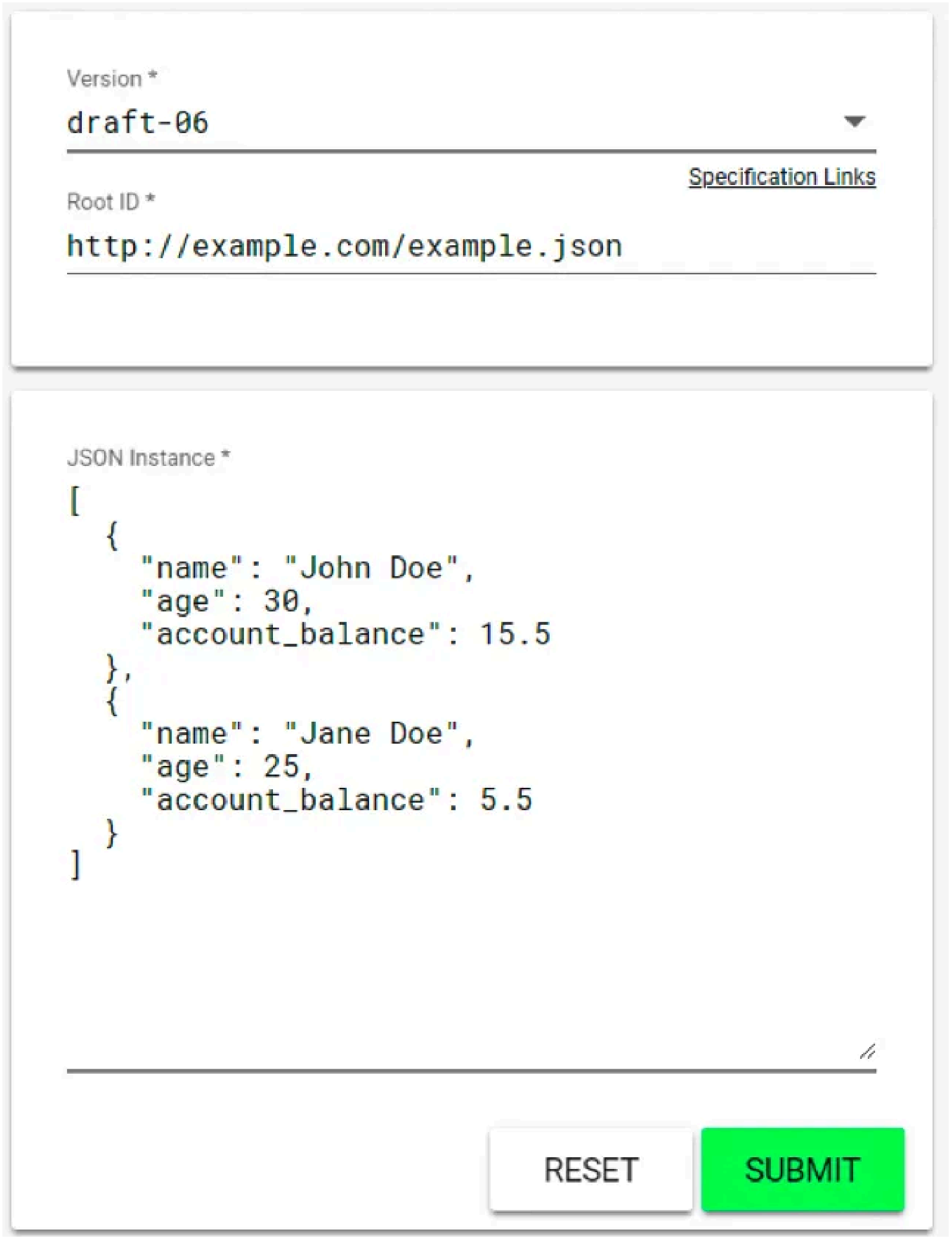
```
[
  {
    "name":"John Doe",
    "age":30,
    "account_balance":15.50
  },
  {
    "name":"Jane Doe",
    "age":25,
    "account_balance":5.50
  }
]
```

`http://localhost:8888/APITesting/Service/getStudent.json`

```
{
  "name": "John Doe",
  "age": 30,
  "account_balance": 15.50,
  "address":"123 Street Name"
}
```

Postman currently supports tv4 for schema validation. We will use https://jsonschema.net/#/ for generating the schema.

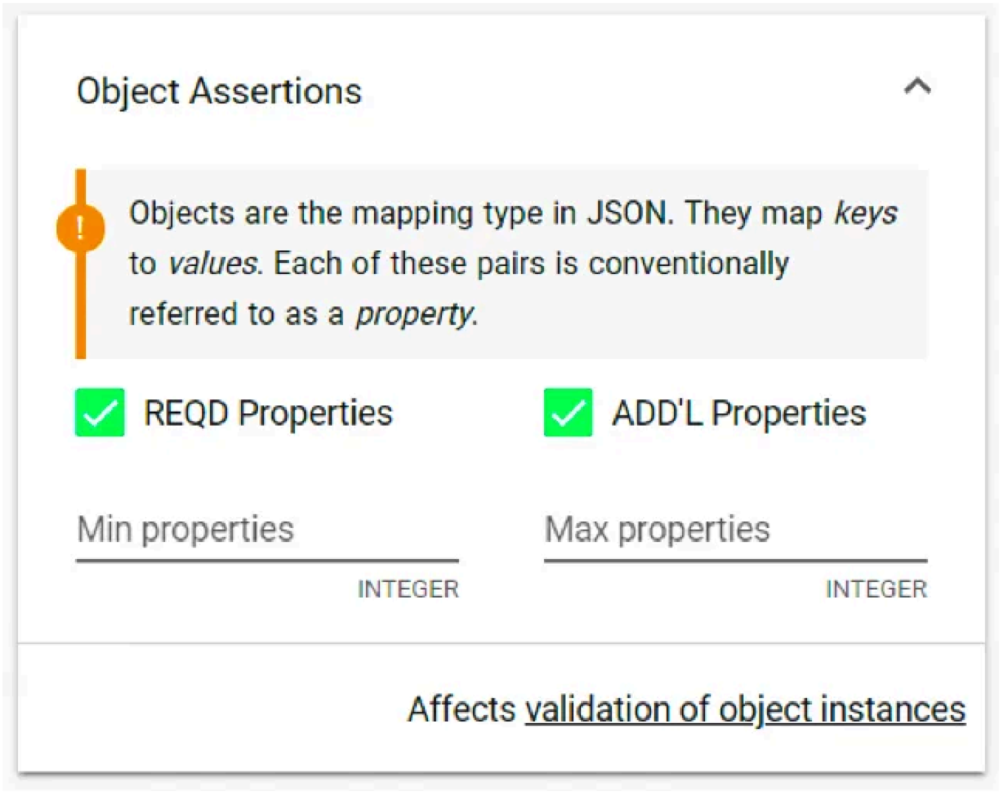
Copy and Paste the JSON doc into the JSON Instance. You can change the `Root ID`, we are going to leave that as is.



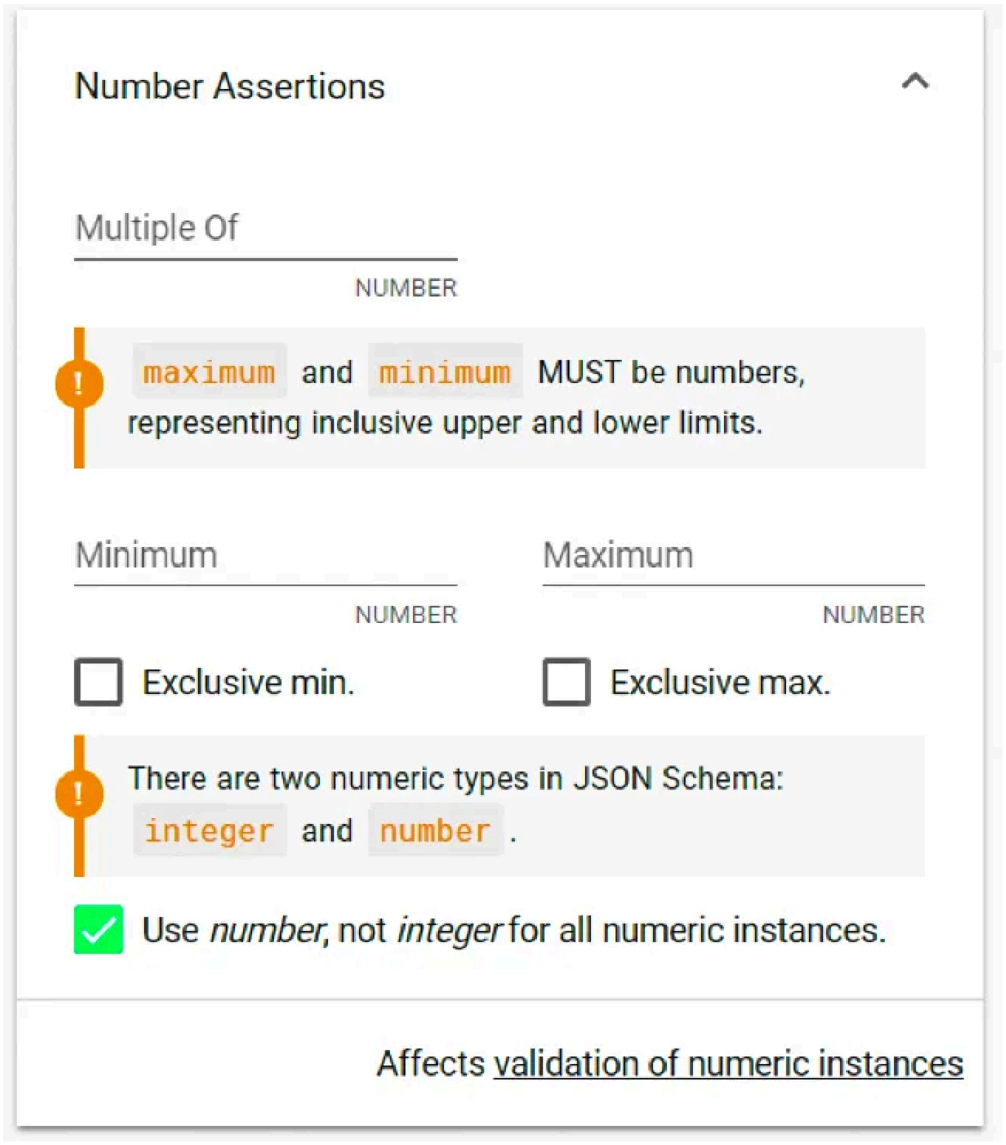
http://example.com/example.json

```
[
  {
    "name": "John Doe",
    "age": 30,
    "account_balance": 15.5
  },
  {
    "name": "Jane Doe",
    "age": 25,
    "account_balance": 5.5
  }
]
```

Then scroll down and expand `Object Assertions` and check `REQD Properties`. We do this so that every element in the JSON becomes mandatory.


Object Assertions
Objects are the mapping type in JSON. They map *keys* to *values*. Each of these pairs is conventionally referred to as a *property*.
✓ REQD Properties
✓ ADD'L Properties
Min properties
INTEGER
Max properties
INTEGER
Affects validation of object instances

Expand `Number Assertions` and check `Use number, not integer for all numeric instances`.


Number Assertions

Multiple Of
NUMBER

`maximum` and `minimum` MUST be numbers, representing inclusive upper and lower limits.

Minimum
NUMBER

Maximum
NUMBER

☐ Exclusive min.
☐ Exclusive max.

There are two numeric types in JSON Schema: `integer` and `number`.

✓ Use *number*, not *integer* for all numeric instances.

Affects validation of numeric instances

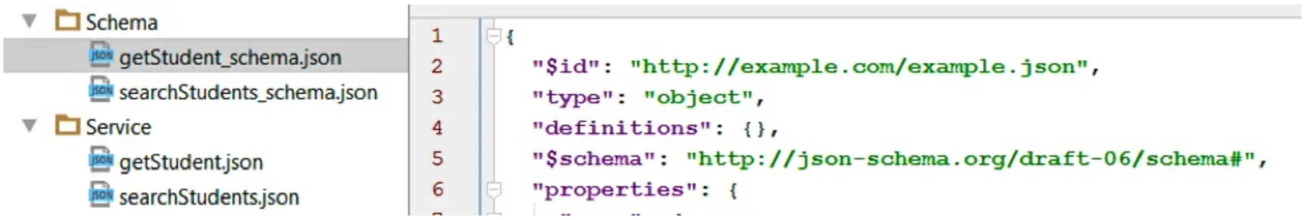
Feel free to play with other settings as needed for your app.

Next Click on Submit and generate the schema. Here is the schema which was generated.

```
{
  "$id": "http://example.com/example.json",
  "type": "array",
  "definitions": {},
  "$schema": "http://json-schema.org/draft-06/schema#",
  "items": {
    "$id": "http://example.com/example.json/items",
    "type": "object",
    "properties": {
      "name": {
        "$id": "http://example.com/example.json/items/properties/name",
        "type": "string",
        "title": "The Name Schema",
        "description": "An explanation about the purpose of this instance.",
        "default": "",
        "examples": [
          "John Doe"
        ]
      },
      "age": {
        "$id": "http://example.com/example.json/items/properties/age",
        "type": "number",
        "title": "The Age Schema",
        "description": "An explanation about the purpose of this instance.",
        "default": 0,
        "examples": [
          30
        ]
      },
      "account_balance": {
        "$id":
"http://example.com/example.json/items/properties/account_balance",
        "type": "number",
        "title": "The Account_balance Schema",
        "description": "An explanation about the purpose of this instance.",
        "default": 0,
        "examples": [
          15.5
        ]
      }
    },
    "required": [
      "name",
      "age",
      "account_balance"
    ]
  }
}
```

We will store the schema in our web/app server so that it can be accessed by Postman during testing. I am using MAMP since it provides the webserver. You can use brackets.io for local testing, however during QA you need to have these schemas as part of the application/web server itself, so that from any process the test scripts can be executed.

Here I simplly save the file in local and repeat the process for the 2nd service. Here is how it looks in my local.

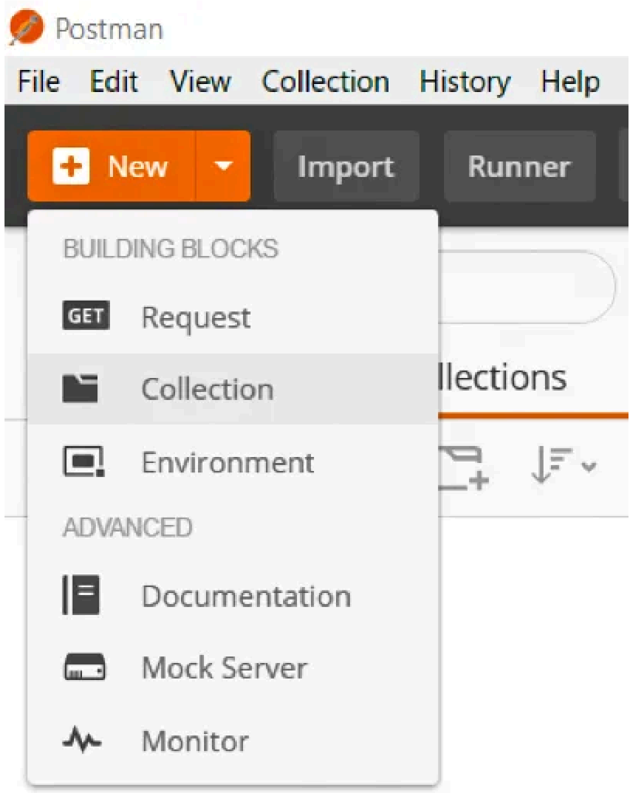


```
{
  "$id": "http://example.com/example.json",
  "type": "object",
  "definitions": {},
  "$schema": "http://json-schema.org/draft-06/schema#",
  "properties": {
```

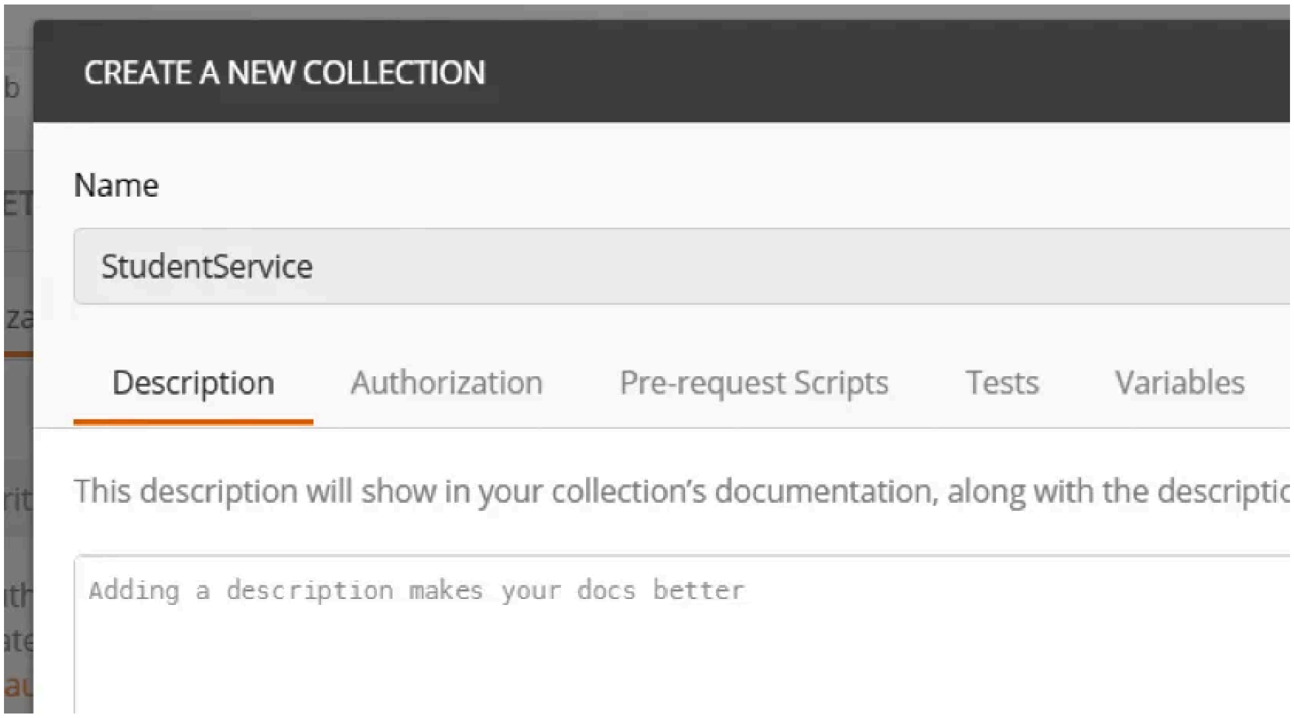
Configure and Test using Postman

Lets configure the Postman using our service endpoints.

First lets start with creation of a new collection.



Name as the collection as Student Service.



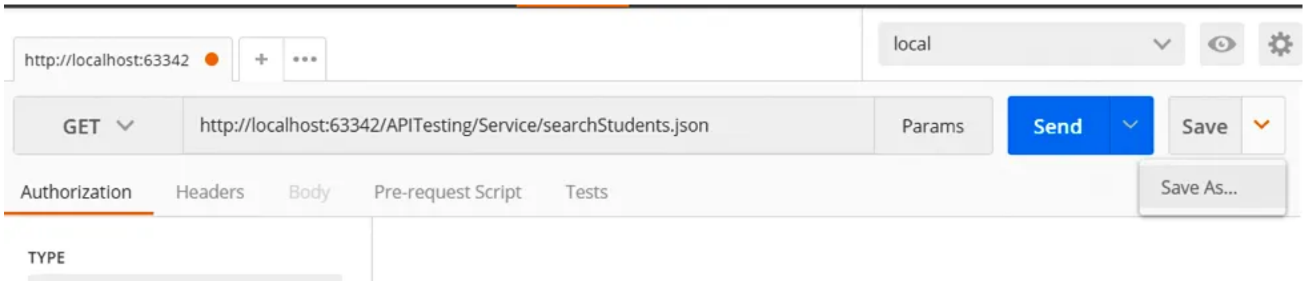
Name

StudentService

This description will show in your collection's documentation, along with the descriptic

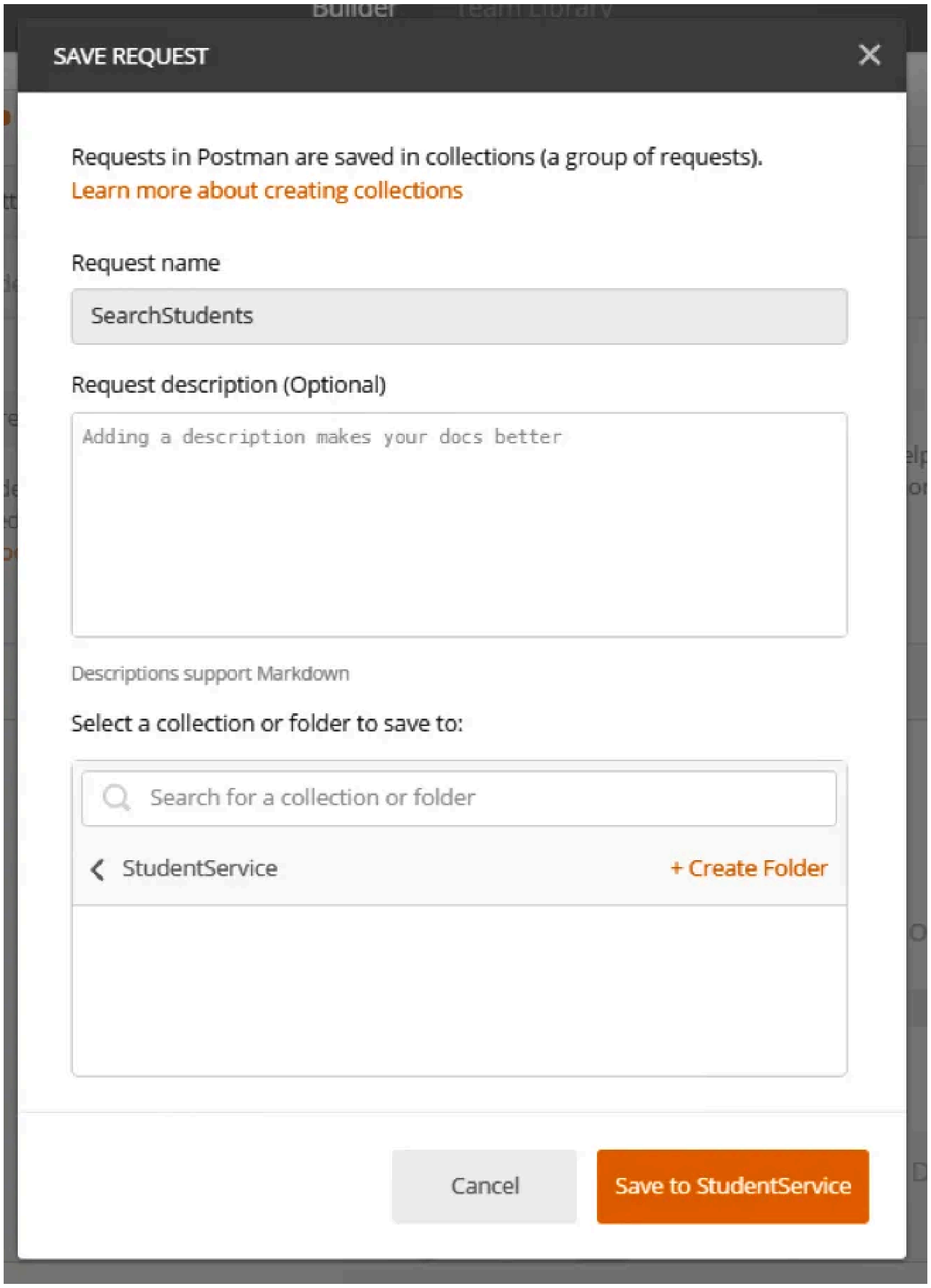
Adding a description makes your docs better

Add the URL of the Student Service and save that as Search Student. Repeate the same for the Get Student Service.



http://localhost:63342/APITesting/Service/searchStudents.json

Save the URL inside Student Service collection.


SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
Learn more about creating collections

Request name

SearchStudents

Request description (Optional)

Adding a description makes your docs better

Descriptions support Markdown

Select a collection or folder to save to:

Search for a collection or folder

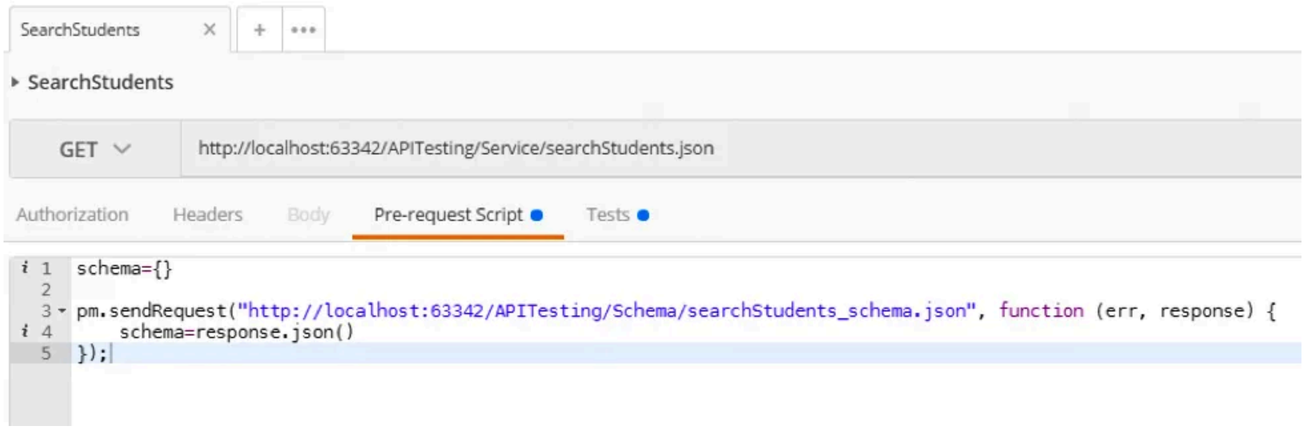
StudentService + Create Folder

Cancel Save to StudentService

Open the pre-request script tab and paste the followng code. Here we are just loading the appropriate schema in postman as a variable.

```
schema={}

pm.sendRequest("http://localhost:8888/Titan/ClaimPayments/paymentAdjustments.json",
function (err, response) {
    schema=response.json()
});
```

Here is how this looks (Ignore the port number).

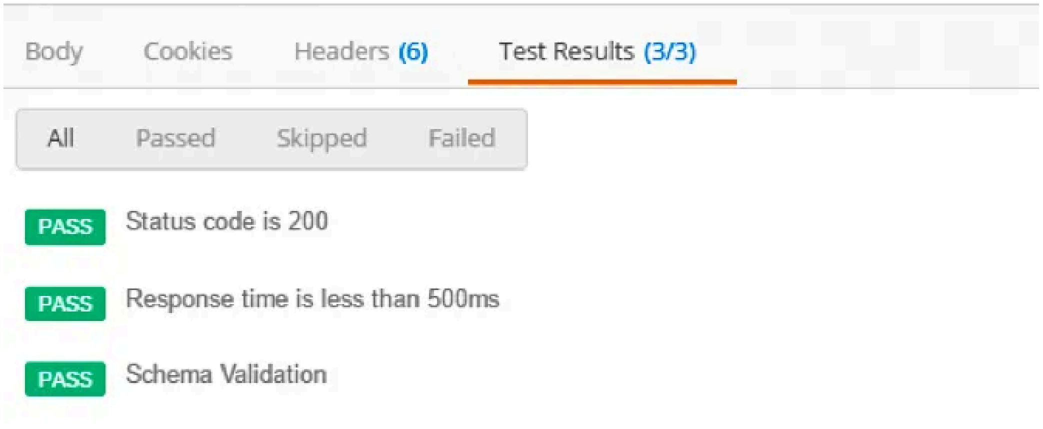


```
schema={}

pm.sendRequest("http://localhost:63342/APITesting/Schema/searchStudents_schema.json", function (err, response) {
    schema=response.json()
});
```

Now, add the following code to the tests tab. Here we have some generic test cases as well. However the 3rd test is for validating the schema. We are using `tv4` and `validateResult` function. In case the JSON is valid, we are not logging anything, otherwise we are logging the error in the console.

```
//Check response code
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

//Check response time
pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});

//Validate Schema
pm.test('Schema Validation', function() {

    var result=tv4.validateResult(JSON.parse(responseBody), schema);

    if(!result.valid){
        console.log(result);
    }

    pm.expect(result.valid).to.be.true;
})
```

If you click on send to execute the API, all three test cases will be successful.



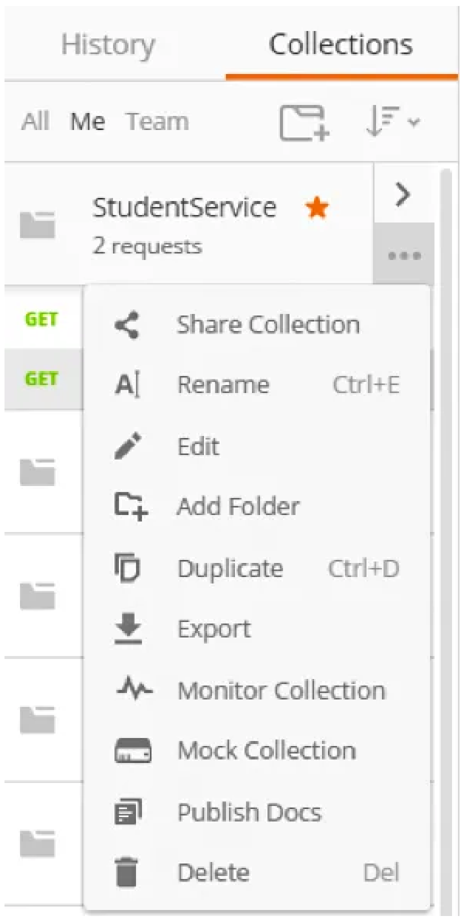
PASS Status code is 200

PASS Response time is less than 500ms

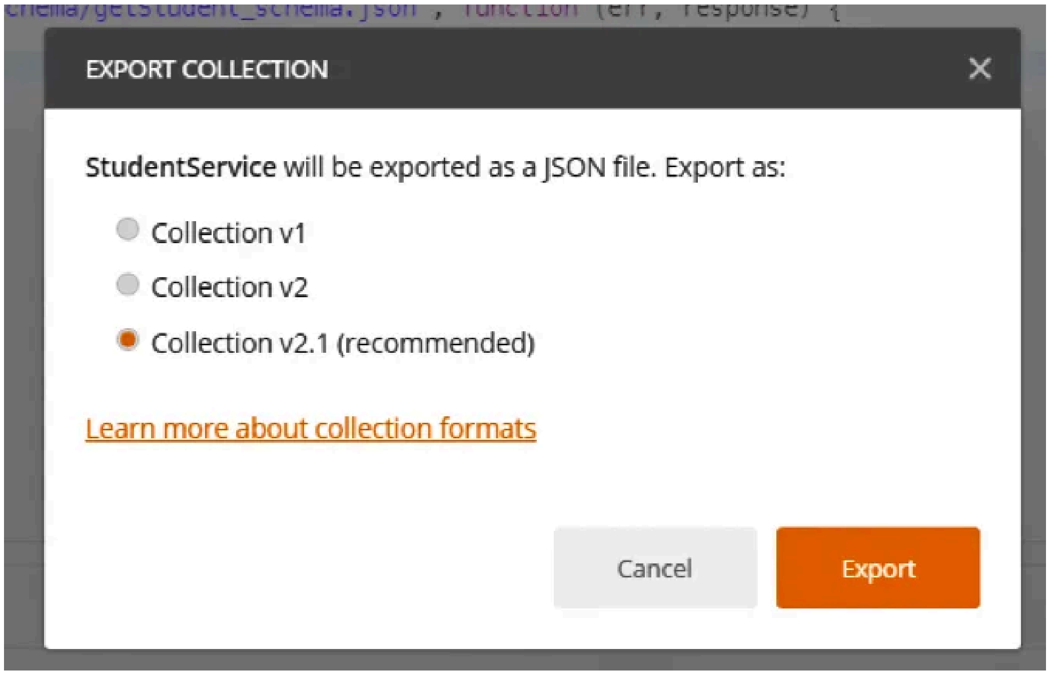
PASS Schema Validation

Run from newman

The final step is to run this from command line. You need to export the collection first. Here is the screen print for that (click on Export).



Select Collection v2.1 and click on export. Save the file in local dir.


EXPORT COLLECTION

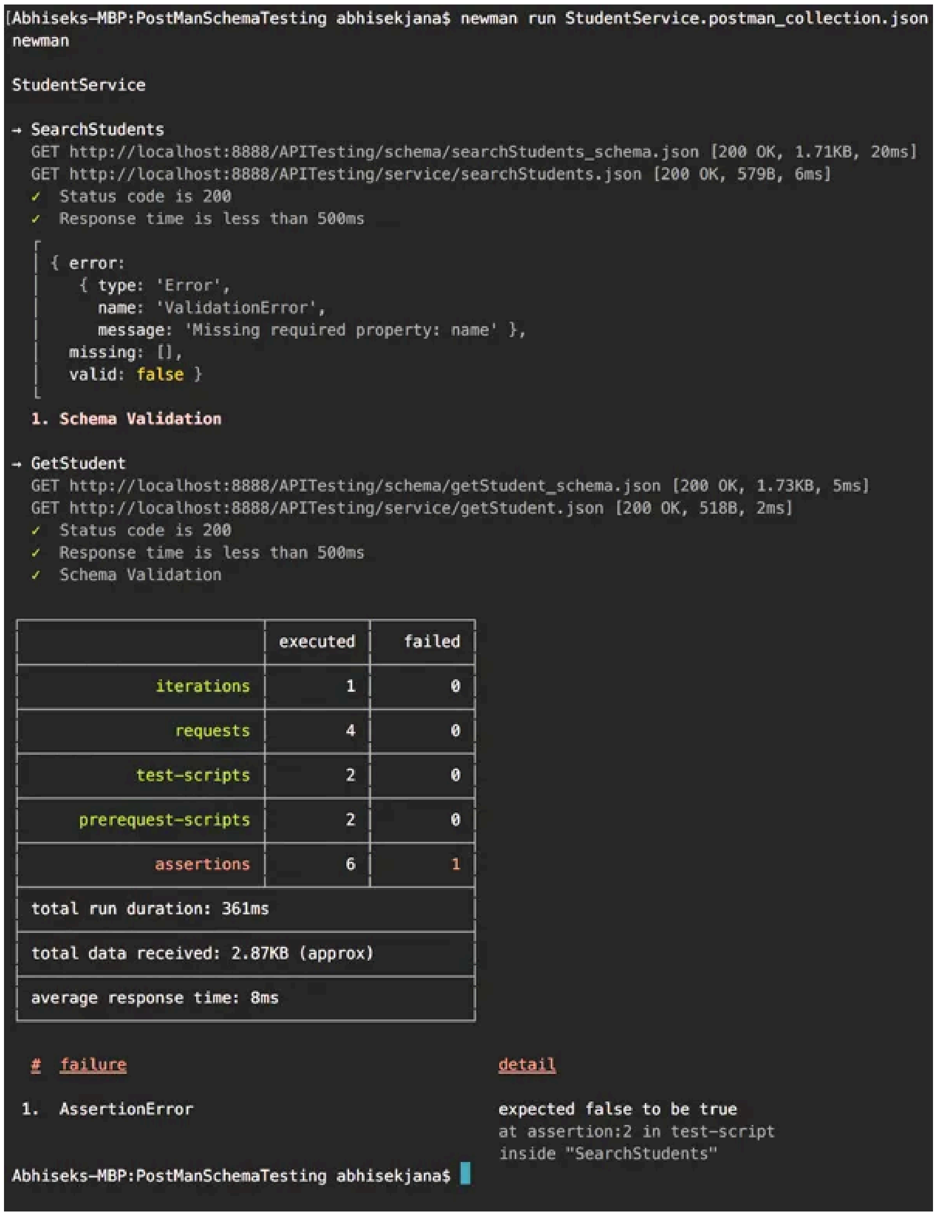
StudentService will be exported as a JSON file. Export as:

( ) Collection v1
( ) Collection v2
[✓ Collection v2.1 (recommended)]

Learn more about collection formats

I assume that you have already installed newman. So the next step would be to execute the same from command line.

`newman run StudentService.postman_collection.json`
or using remote call.
`newman run http://localhost:8888/APITesting/script/StudentService.postman_collection.json`



```
[Abhiseks-MBP:PostManSchemaTesting abhisekjana$ newman run StudentService.postman_collection.json
newman

StudentService

→ SearchStudents
  GET http://localhost:8888/APITesting/schema/searchStudents_schema.json [200 OK, 1.71KB, 20ms]
  GET http://localhost:8888/APITesting/service/searchStudents.json [200 OK, 579B, 6ms]
  ✓  Status code is 200
  ✓  Response time is less than 500ms
  ┌
  │ { error:
  │    { type: 'Error',
  │      name: 'ValidationError',
  │      message: 'Missing required property: name' },
  │   missing: [],
  │   valid: false }
  └
  1. Schema Validation

→ GetStudent
  GET http://localhost:8888/APITesting/schema/getStudent_schema.json [200 OK, 1.73KB, 5ms]
  GET http://localhost:8888/APITesting/service/getStudent.json [200 OK, 518B, 2ms]
  ✓  Status code is 200
  ✓  Response time is less than 500ms
  ✓  Schema Validation

┌─────────────────────────┬──────────┬──────────┐
│                         │ executed │   failed │
├─────────────────────────┼──────────┼──────────┤
│              iterations │        1 │        0 │
├─────────────────────────┼──────────┼──────────┤
│                requests │        4 │        0 │
├─────────────────────────┼──────────┼──────────┤
│            test-scripts │        2 │        0 │
├─────────────────────────┼──────────┼──────────┤
│      prerequest-scripts │        2 │        0 │
├─────────────────────────┼──────────┼──────────┤
│              assertions │        6 │        1 │
├─────────────────────────┴──────────┴──────────┤
│ total run duration: 361ms                     │
├───────────────────────────────────────────────┤
│ total data received: 2.87KB (approx)          │
├───────────────────────────────────────────────┤
│ average response time: 8ms                    │
└───────────────────────────────────────────────┘

  #  failure                                 detail

 1.  AssertionError                          expected false to be true
                                             at assertion:2 in test-script
                                             inside "SearchStudents"

Abhiseks-MBP:PostManSchemaTesting abhisekjana$
```

I have made an intentional mistake in the JSON, so you can see the error log get displayed here. This provides all the detailed information.

Next Step:

There are many enhancement you can implement, such as –

- Have a common test script at the Collection level and just define the environment values to load proper schemas. This will element addition of same script in each service tests.
- Add additional validation/test cases using javascript
- Integrate this to the build process and export a report
- Finally, both Postman and newman has many advanced features which you can use.

Conclusion:

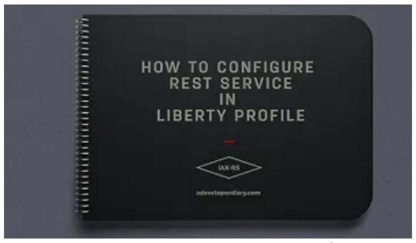
I hope this article really shows you How to Automate REST API Schema Validation testing using Postman. I was looking for the same type of solution and after going through many articles I was able to use this process. I am also very new to Postman, so as I learn new tricks and features I will add it here.

Feel free to post any question on this below.

Related

How to connect to TitanDB using Gremlin API
In "DataBase"

How to create RESTFul Webservices using Spring Boot
In "Spring Boot"



How to configure REST Service (JAX-RS) in IBM Liberty Profile
In "REST"

Filed Under: REST | Tagged With: api, automated, JSON, newman, postman, REST, RESTFul, schema, testing, tv4, unit testing

Subscribe to stay in loop

* indicates required

Email Address *

Subscribe

Trackbacks

Confluence: Ejazah OTA Flights says:
April 7, 2020 at 8:53 am Edit

API testing

API stability – is a first-priority issue for the

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment