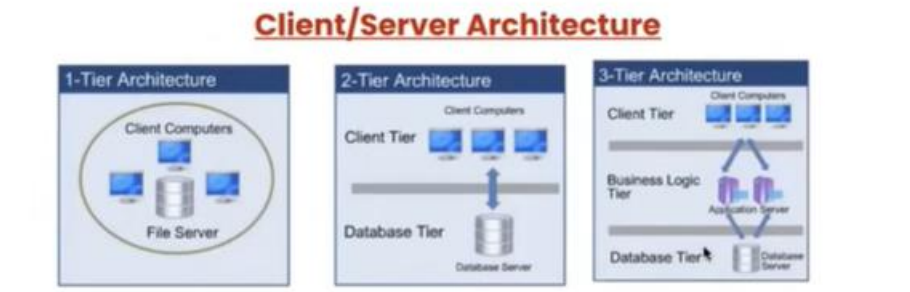
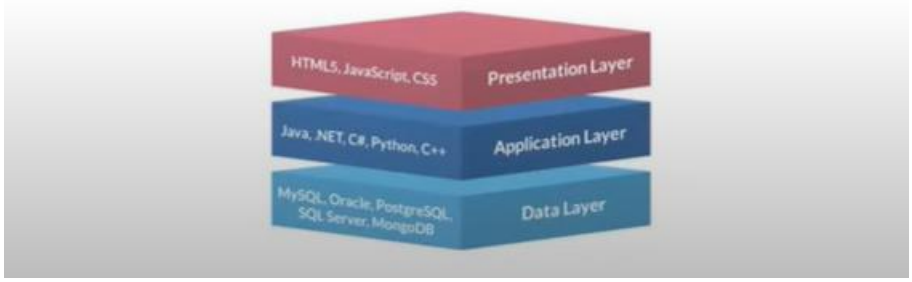
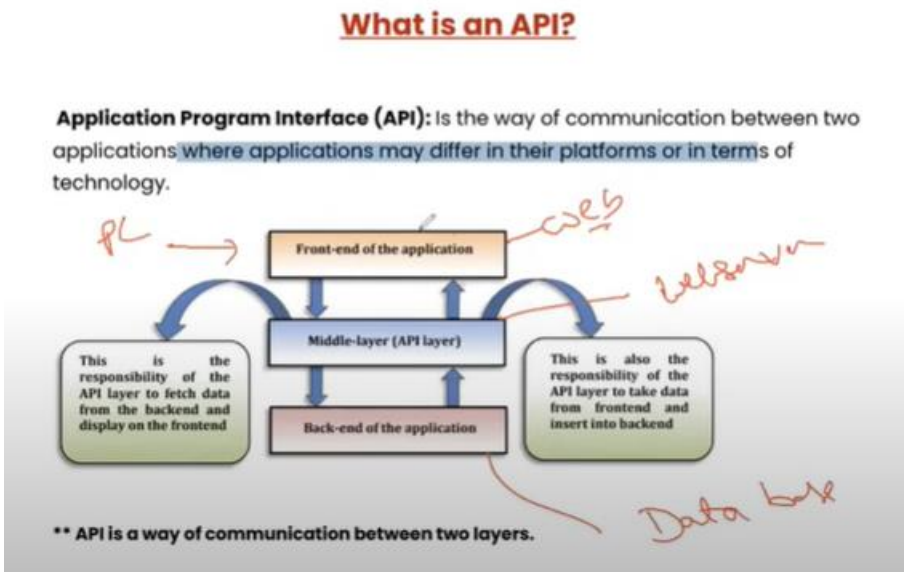
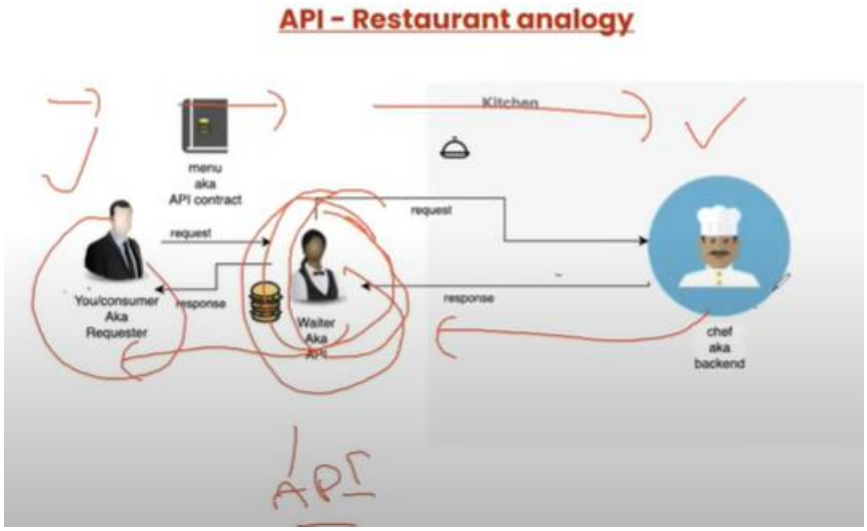
API Testing: Application Program Interface

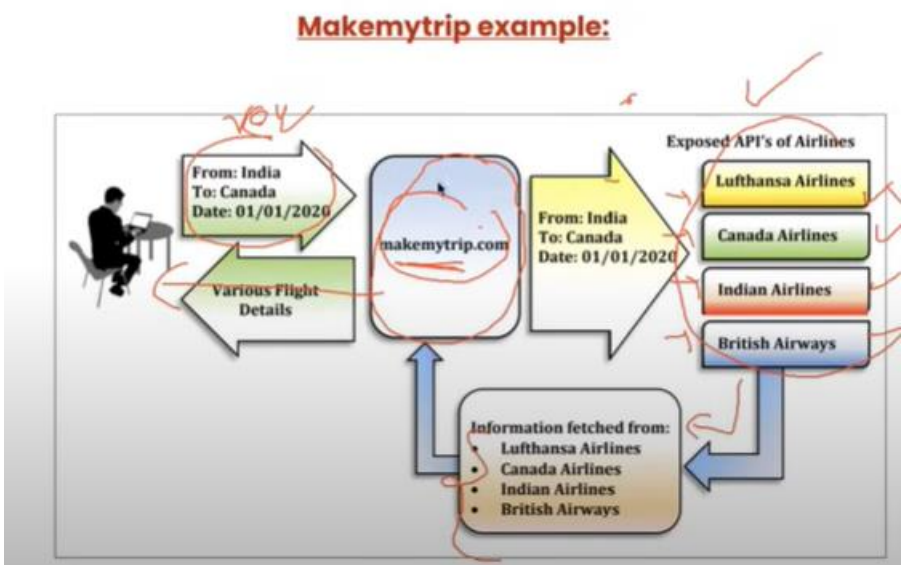
It is way of communication between 2 applications where application may differ in their platforms or in terms of technology.











Types of API:

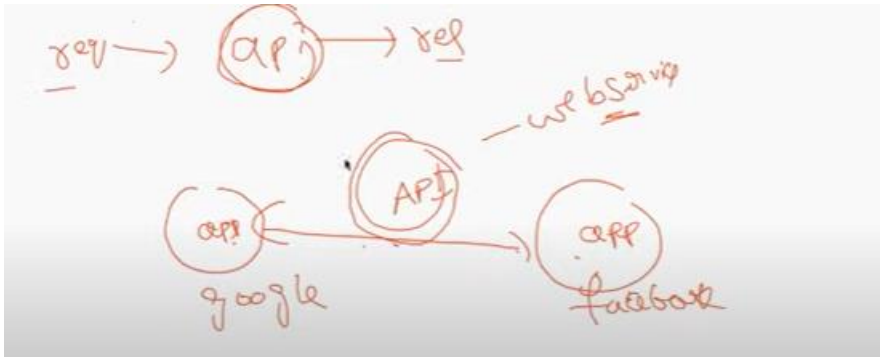
There re 2 types of API's.

1) Simple Object Access Protocol (SOAP).
2) REST (Representation State Transfer).

Both are the web services.

SOAP => used by old applications.

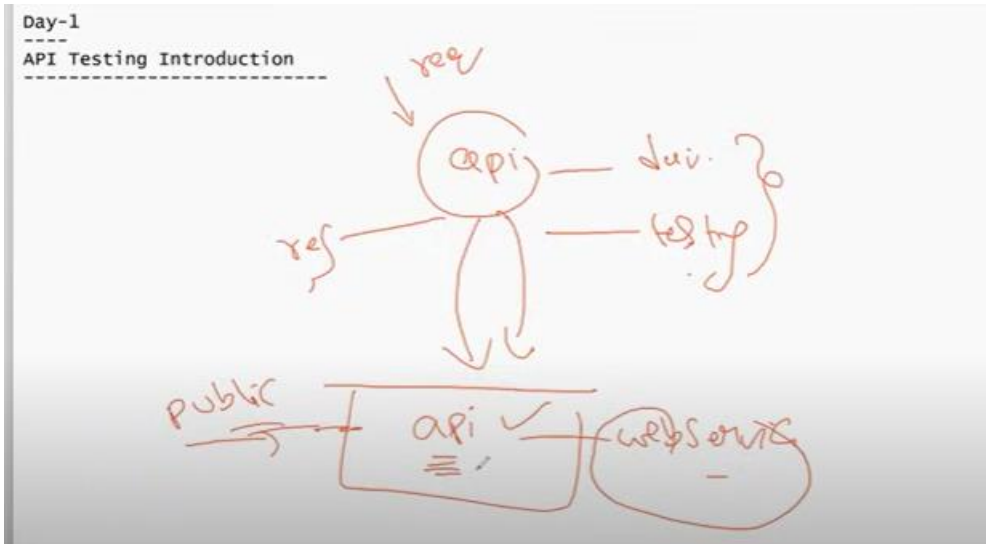
REST => used by new/current applications.



All Web Services are API. But all API are not Web Service.

When API is put in internet then it is called as Web Service.

For development & testing we use Api. And once its available to public over internet its called Webservice.



Important: API Vs Webservice

Types Of API

There are two types of API's,

1. Simple Object Access Protocol (SOAP)
2. REST (Representational State Transfer).

Both are the web services.

API Vs Webservice

- Web Service is an API wrapped in HTTP.
- All Web Services are API but APIs are not Web Services.
- A Web Service needs a network while an API doesn't need a network for its operation.

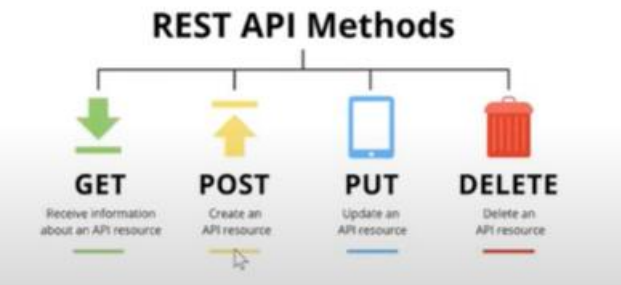
Rest API methods: http request

Get: we get data from the server.

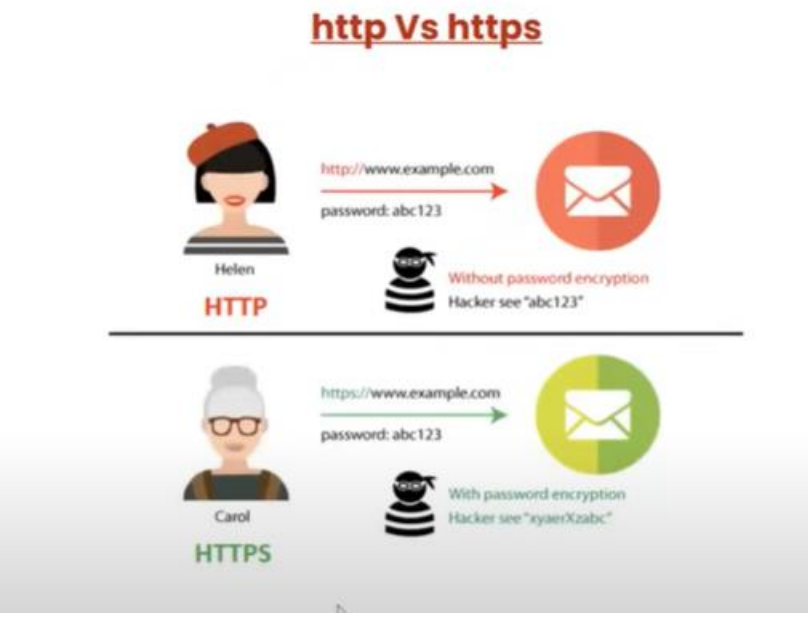
Post: we are sending data to server to store.

Put: we can create/edit/update data in the server.

Delete: we delete the data from server.



HTTP Vs HTTPS



Terminology:

Terminologies

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

URN – Uniform Resource Name

Payload:

Feature & Resource

'**Feature**' is the term used in manual testing to test some functionality and similarly '**Resource**' is the term used in API Automation testing referring some functionality.

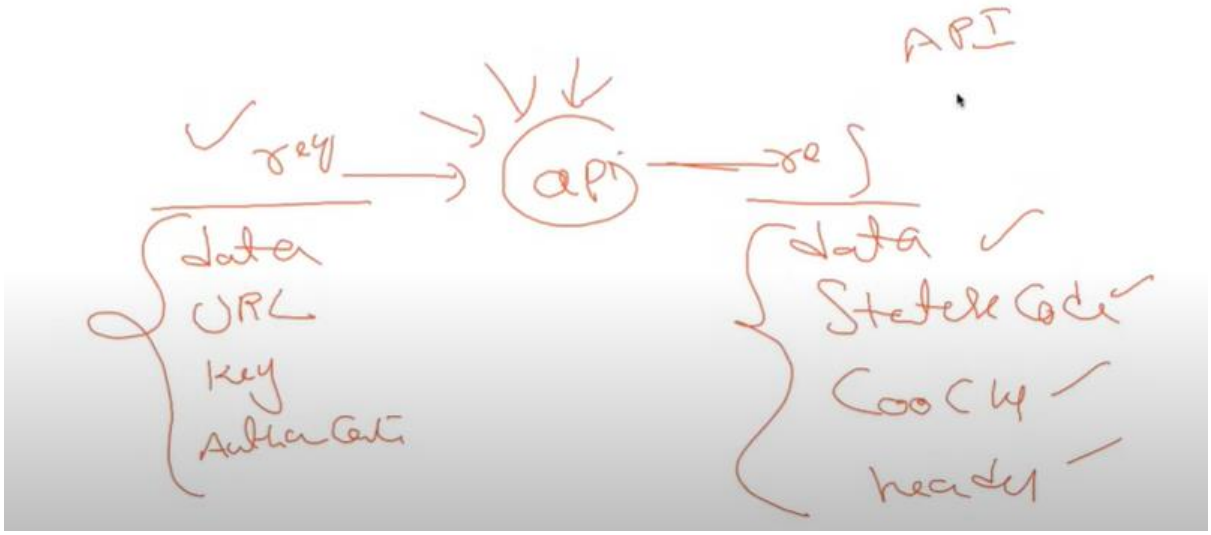
Payload

payload means **body in the HTTP request and response message**.

- Request Payload
- Response Payload

Ex: https://reqres.in/

Ex Get: https://reqres.in/api/users?page=2 => https://reqres.in/api/users?page=2



Rest API Examples

1. Payment Gateways
2. Google Maps
3. Twitter
4. Facebook
5. Linked-In
6. GitHub

Postman – API testing

Desktop/web

Workspace: area where we maintain files and saved.

Workspace – create workspace, rename, delete.

Creating collection = contains number of folders and http requests. Create, rename, delete, run the collection.

We can create any number of collections under workspace.

Request ----→ API -------→ Response

http Request:

Get => retrieve the resource from database.

Post => create resource on database.

Put => update existing resource on database.

Patch => update partial details of resource.

Delete => delete existing resource from database.

Sample APIs: https://reqres.in/

Request Type	URI	Request Payload	Response Payload	Status Code
GET	https://reqres.in/api/users?page=2	NA	Returns list of users in a page	200
POST	https://reqres.in/api/users	{ "name": "pavan", "job": "trainer" }	{ "id": "599", "createdAt": "2018-07-07T05:43:53.310Z" }	201
PUT	https://reqres.in/api/users/599	{ "name": "pavan", "job": "engineer" }	{ "updatedAt": "2022-07-16T05:08:14.041Z" }	200
DELETE	https://reqres.in/api/users/599	na	na	204

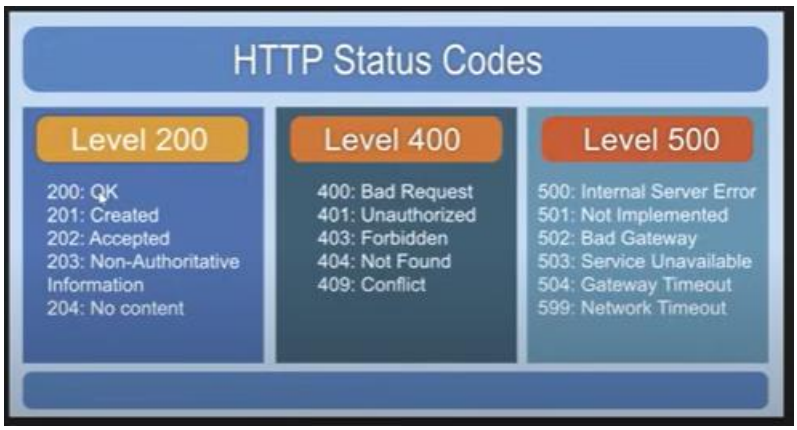
Validations:

1) Status code
2) Time
3) Size data
4) Response body(json/xml)
5) Cookies
6) Headers

HTTP Status code:

3 levels

1) 200
2) 400
3) 500



How to create own API: JSON

Create our own API's

Steps:

1) NodeJS
2) Npm-node package manager
   node - -version
   npm --version

3) Json-server
4) Install json-server:
   Run below command in cmd/terminal
   npm install -g json-server

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Rutuja> npm install -g json-server

added 57 packages in 17s

18 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New minor version of npm available! 10.7.0 -> 10.8.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.2
npm notice To update run: npm install -g npm@10.8.2
npm notice
PS C:\Users\Rutuja>
```

5) Create students.json file with following data.

```
{
    "students":[
        {
            "id": 1,
            "name": "John",
            "location": "India",
            "phone": "1234567890",
            "courses": [
                "Java",
                "Selenium"
            ]

        },
        {
            "id": 2,
            "name": "Kim",
            "location": "US",
            "phone": "2345678901",
            "courses": [
                "Python",
                "Appium"
```

```
            ]
        },
        {
            "id": 3,
            "name": "Smith",
            "location": "Canada",
            "phone": "3456789012",
            "courses": [
                "C#",
                "RestAPI"
            ]
        }
]
}
```

6) Run using command ‘json-server students.json’

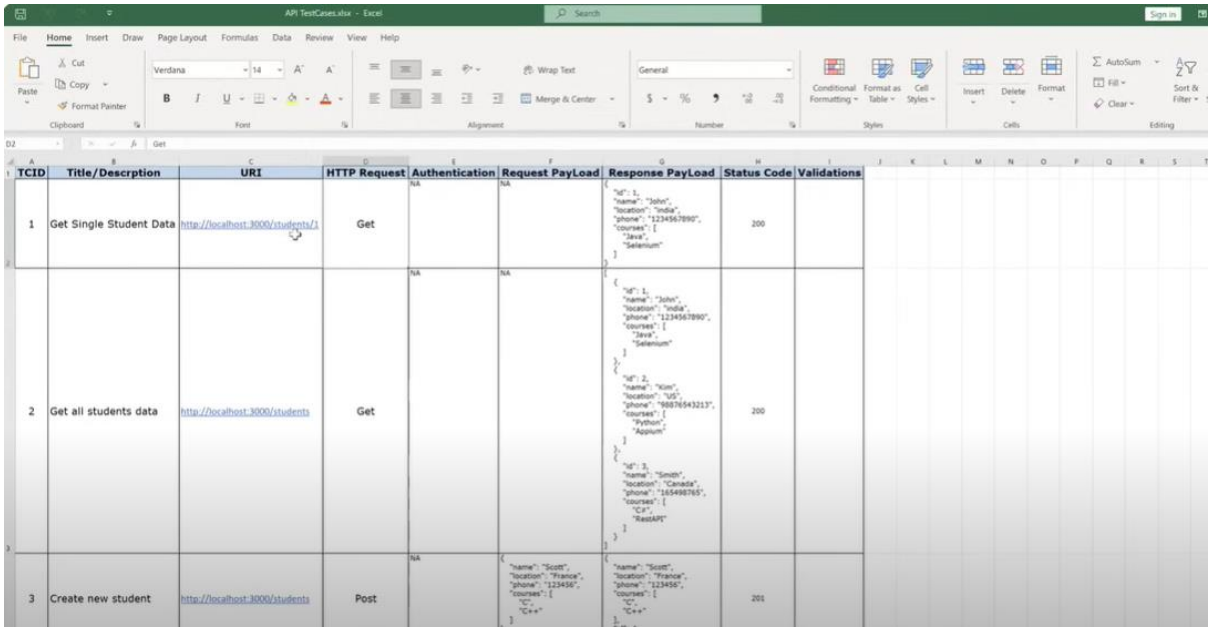
```
D:\Automation\API Testing\API_Testing>json-server students.json
JSON Server started on PORT :3000
Press CTRL-C to stop
Watching students.json...

( ~^ ᗢ ^~ )

Index:
http://localhost:3000/

Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:3000/students
```

7) http://localhost:3000/students

localhost:3000/students

Pretty-print ☑


```
[
  {
    "id": "1",
    "name": "John",
    "location": "India",
    "phone": "1234567890",
    "courses": [
      "Java",
      "Selenium"
    ]
  },
  {
    "id": "2",
    "name": "Kim",
    "location": "US",
    "phone": "2345678901",
    "courses": [
      "Python",
      "Appium"
    ]
  },
  {
    "id": "3",
    "name": "Smith",
    "location": "Canada",
    "phone": "3456789012",
    "courses": [
      "C#",
      "RestAPI"
    ]
  }
]
```

8) 

TCID	Title/Descrption	URI	HTTP Request	Authentication	Request PayLoad	Response PayLoad	Status Code	Validations
1	Get Single Student Data	http://localhost:3000/students/1	Get	NA	NA	{ "id": 1, "name": "John", "location": "India", "phone": "1234567890", "courses": ["Java", "Selenium"] }	200	
2	Get all students data	http://localhost:3000/students	Get	NA	NA	[{ "id": 1, "name": "John", "location": "India", "phone": "1234567890", "courses": ["Java", "Selenium"] }, { "id": 2, "name": "Kim", "location": "US", "phone": "98876543213", "courses": ["Python", "Appium"] }, { "id": 3, "name": "Smith", "location": "Canada", "phone": "1654987654", "courses": ["C#", "RestAPI"] }]	200	
3	Create new student	http://localhost:3000/students	Post	NA	{ "name": "Scott", "location": "France", "phone": "123456", "courses": ["C", "C++"] }	{ "name": "Scott", "location": "France", "phone": "123456", "courses": ["C", "C++"] },	201	

JSON: Java Script Object Notation

Key value pair

Key: value

JSON Data Types:

1) Number
2) String
3) Boolean
4) Null
5) Object
6) Array

```
{
"name": "John"
}
```

Data Types:

JSON - Syntax

- Data should be in name/value pairs
- Data should be separated by commas
- Curly braces should hold objects
- Square brackets hold arrays

```
{
  "student": [

    {
       "id":"01",
       "name": "Tom",
       "lastname": "Price"
    },

    {
       "id":"02",
       "name": "Nick",
       "lastname": "Thameson"
    }
  ]
}
```

Data Types

- **String**
- Strings in JSON must be written in double quotes.
- Example:

```
{ "name":"John" }
```

- **Numbers**
- Numbers in JSON must be an integer or a floating point.
- Example:

```
{ "age":30 }
```

- **Object**
- Values in JSON can be objects.
- Example:

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

Key is always included in “ “ quotations.

{

“firstname”:”John”,

“secondname”:null,

“age”:30,

“phone”:[12345,67890],

“status”:true

}

Eg: Student data

Student – sid, sname, grad

{

“students”:[

{

“sid”=101,

“sname”=”John”,

“grad”=”A”

},

{

“sid”=102,

“sname”=”Mark”,

“grad”=”B”

}

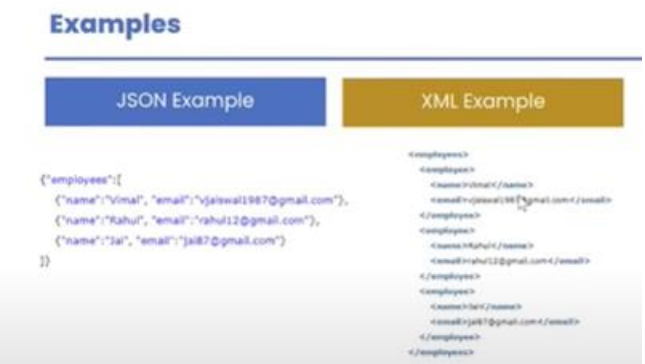
]

}

JSON Path:

students[0].sname --→ John

students[1].sid ---→102



JSON Object & JSON Array:

JSON Object:

- JSON Object with Strings

```
{
"name":    "Scott",
"email":   "scottjaiswal1987@gmail.com"
}
```

- JSON Object with Numbers

```
{
"integer": 34,
"fraction": .2145,
"exponent": 6.61789e+0
}
```

- JSON Object with Booleans

```
{
"first": true,
"second": false
}
```

- JSON Nested Object

```
{
  "firstName": "Scott",
  "lastName": "Jaiswal",
  "age": 27,
  "address" : {
    "streetAddress": "Plot-6, Mohan Nagar",
    "city": "Hyderabad",
    "state": "TL",
    "postalCode": "500090"
  }
}
```

JSON Array

- JSON array represents ordered list of values.
- JSON array can store multiple values. It can store string, number, boolean or object in JSON array.
- In JSON array, values must be separated by comma.
- The [(square bracket) represents JSON array.

Capture & Validate JSON Path

https://jsonpathfinder.com/

https://jsonpath.com/

JSON Path Finder: https://jsonpathfinder.com/

JSON Path Verify: https://jsonpath.com/

API Response Validation | Different types of Assertions:

Response Validation:

1) **Status code**
2) **Headers**
3) **Cookies**
4) **Response time**
5) **Response body**

Assertion – validation

Postman – library

Functions: written in JavaScript

Function types/can be written in 2 ways:

1) Normal function: written using normal keyword i.e. function().
2) Arrow function: written using arrow i.e. () =>.

Library/Framework: Chai

Chai Assertion Library

```
pm.test("Test Name", function()
                {
                  // assertion;
                }
      );
```

```
pm.test("Test Name", () =>
                {
                  // assertion;
                }
      );
```

Here pm = postman.

Testing Status Code:

Testing status codes

Test for the response status code:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
```

If you want to test for the status code being one of a set, include them all in an array and use *one of*

```
pm.test("Successful POST request", () => {
pm.expect(pm.response.code).to.be.oneOf([201,202]);
});
```

Check the status code text:

```
pm.test("Status code name has string", () => {
  pm.response.to.have.status("Created");
});
```

Syntax:

pm.test(“Test Name”, ()=>{

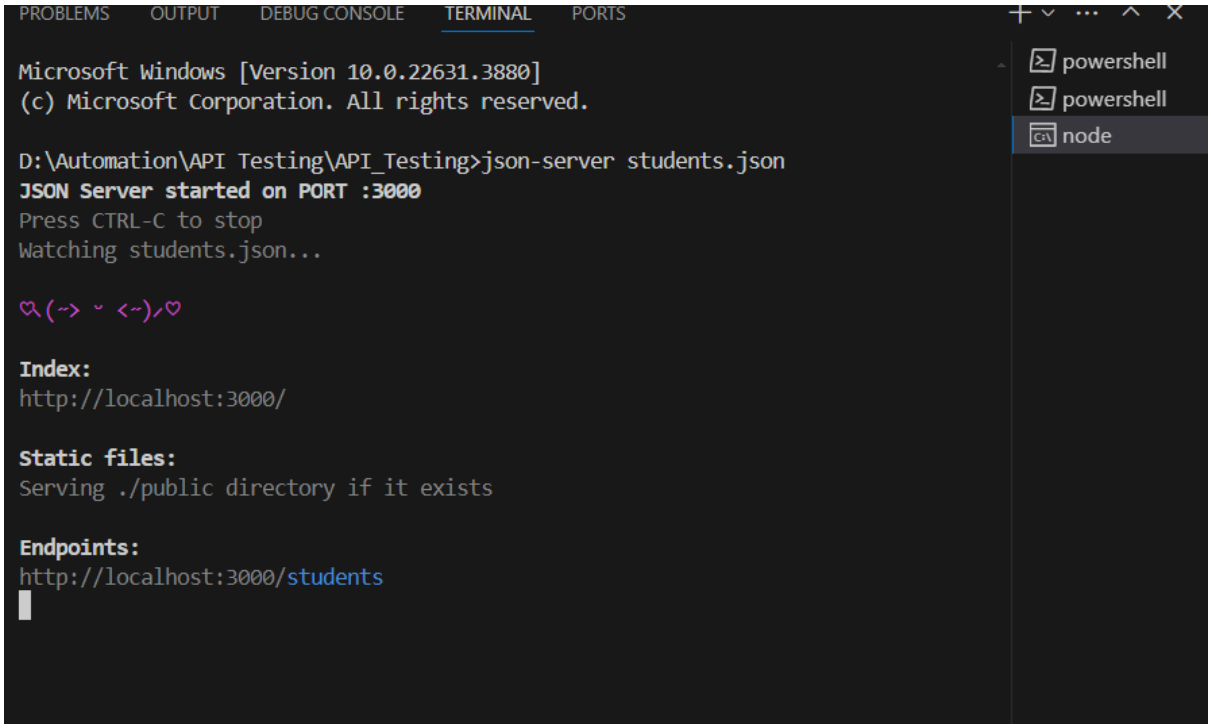
//assertion;

});

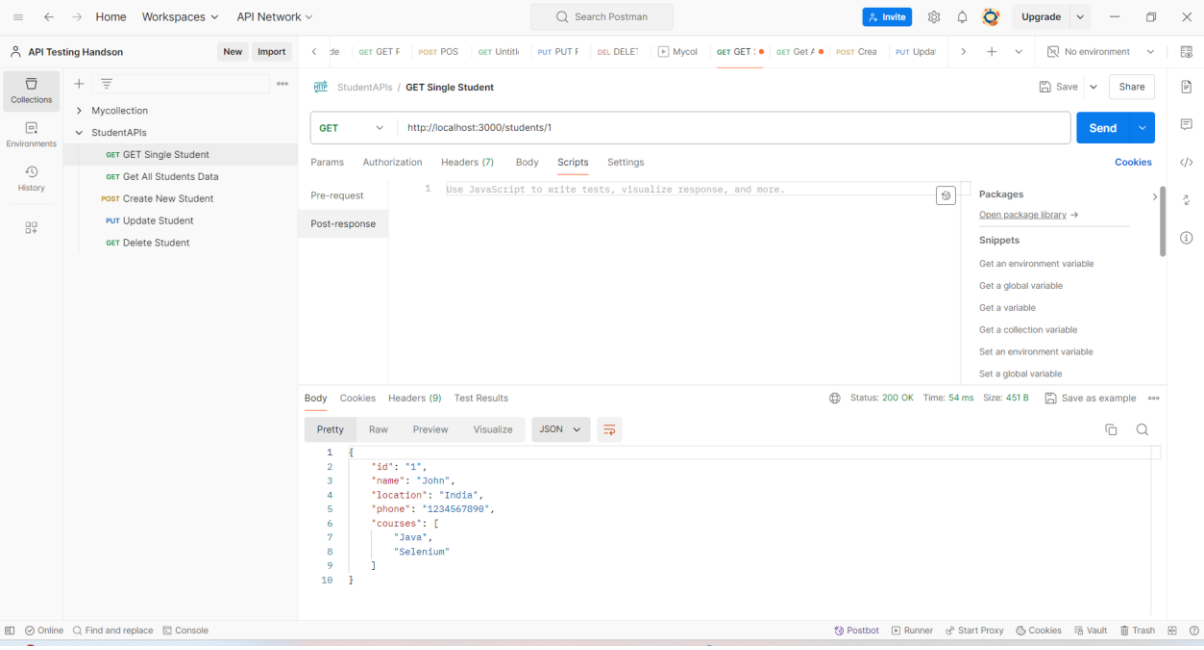
Ex:

pm.test(“Status code is 200”, () => {

```
pm.response.to.have.status(200);

});
```

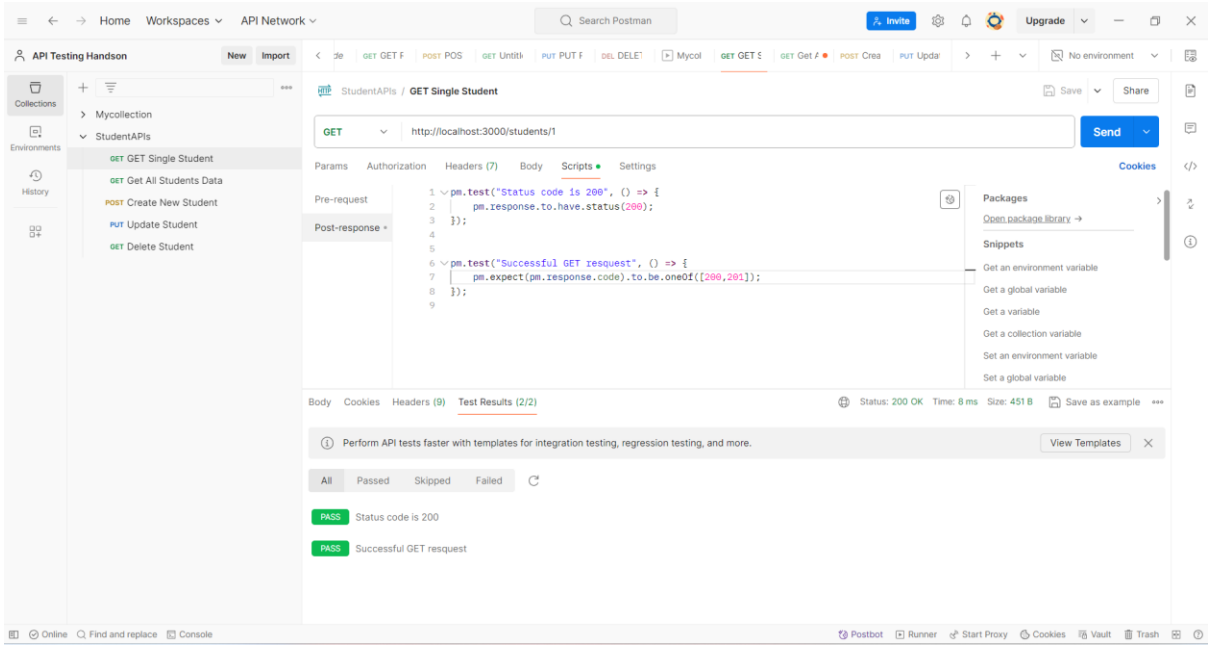


Postman:




```
pm.test("Status code is 200", () => {

   pm.response.to.have.status(200);

});
```

```
pm.test("Successful GET resquest", () => {
   pm.expect(pm.response.code).to.be.oneOf([200,201]);
});
```



Testing status codes

Test for the response status code:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
```

If you want to test for the status code being one of a set, include them all in an array and use *one of*

```
pm.test("Successful POST request", () => {
pm.expect(pm.response.code).to.be.oneOf([201,202]);
});
```

Check the status code text:

```
pm.test("Status code name has string", () => {
  pm.response.to.have.status("Created");
});
```

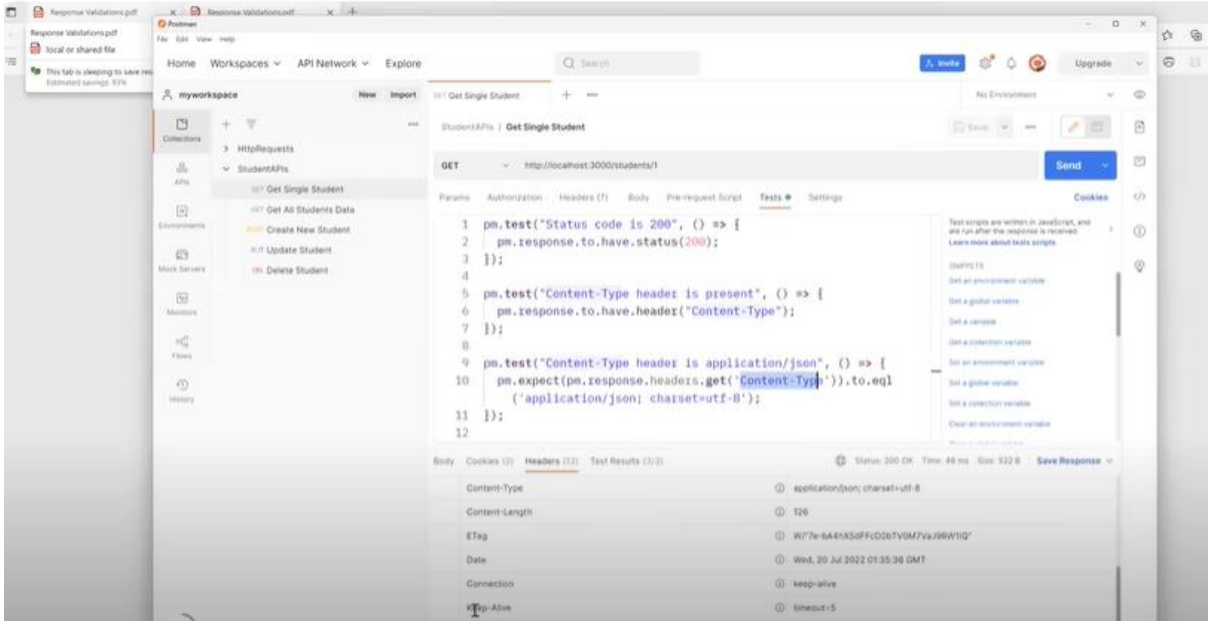
Testing headers

Check that a response header is present:

```
pm.test("Content-Type header is present", () => {
  pm.response.to.have.header("Content-Type");
});
```

Test for a response header having a particular value:

```
pm.test("Content-Type header is application/json", () => {
  pm.expect(pm.response.headers.get('Content-
Type')).to.eql('application/json; charset=utf-8');
});
```



Testing cookies

Test if a cookie is present in the response:

```
pm.test("Cookie 'language' is present", () => {
  pm.expect(pm.cookies.has('language')).to.be.true;
});
```

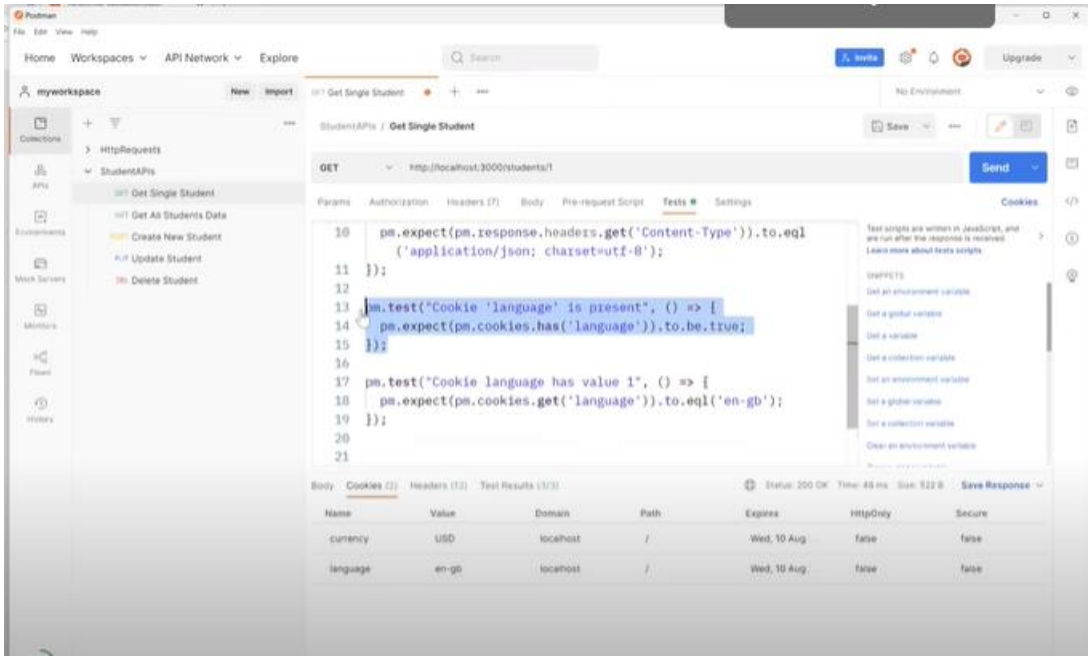
Test for a particular cookie value:

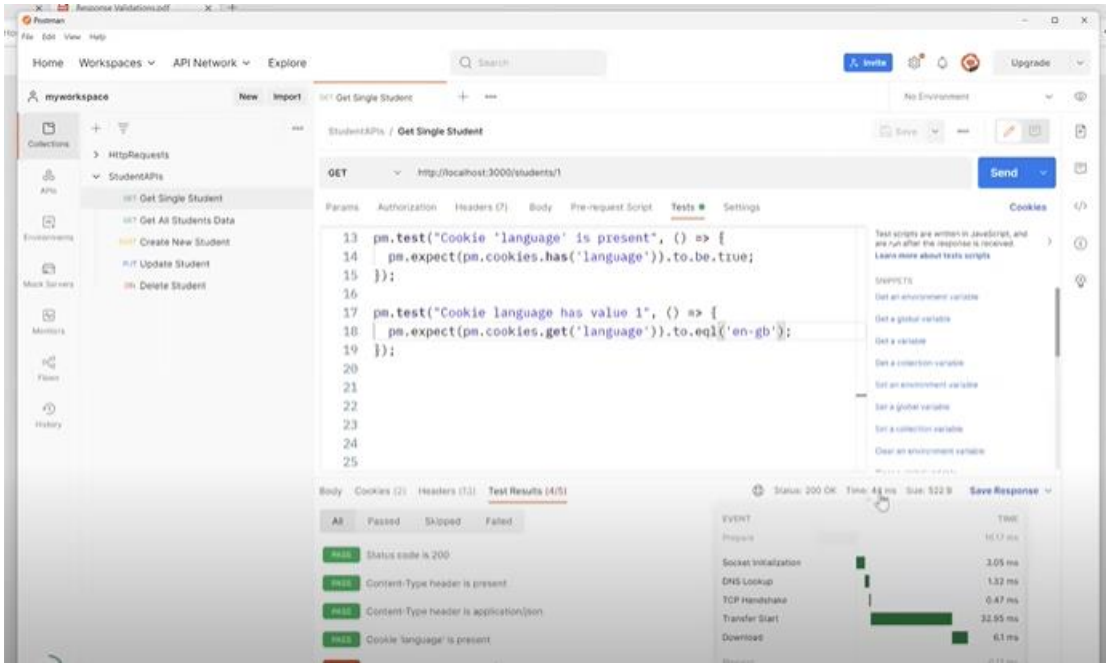
```
pm.test("Cookie language has value 1", () => {
  pm.expect(pm.cookies.get('language')).to.eql('en-gb');
});
```

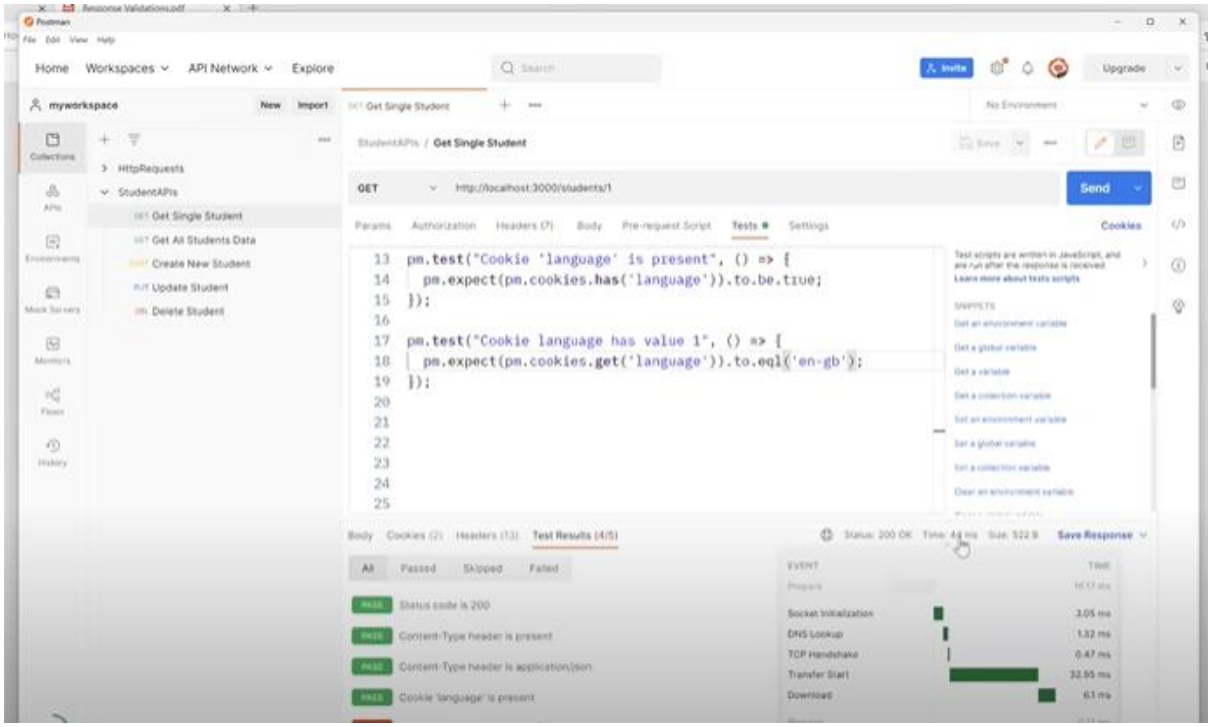
Testing response times

Test for the response time to be within a specified range:

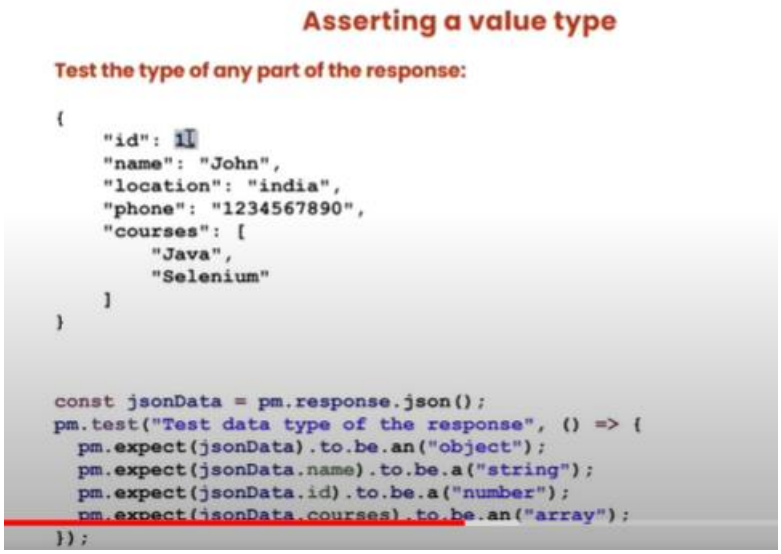
```
pm.test("Response time is less than 200ms", () => {
  pm.expect(pm.response.responseTime).to.be.below(200);
});
```

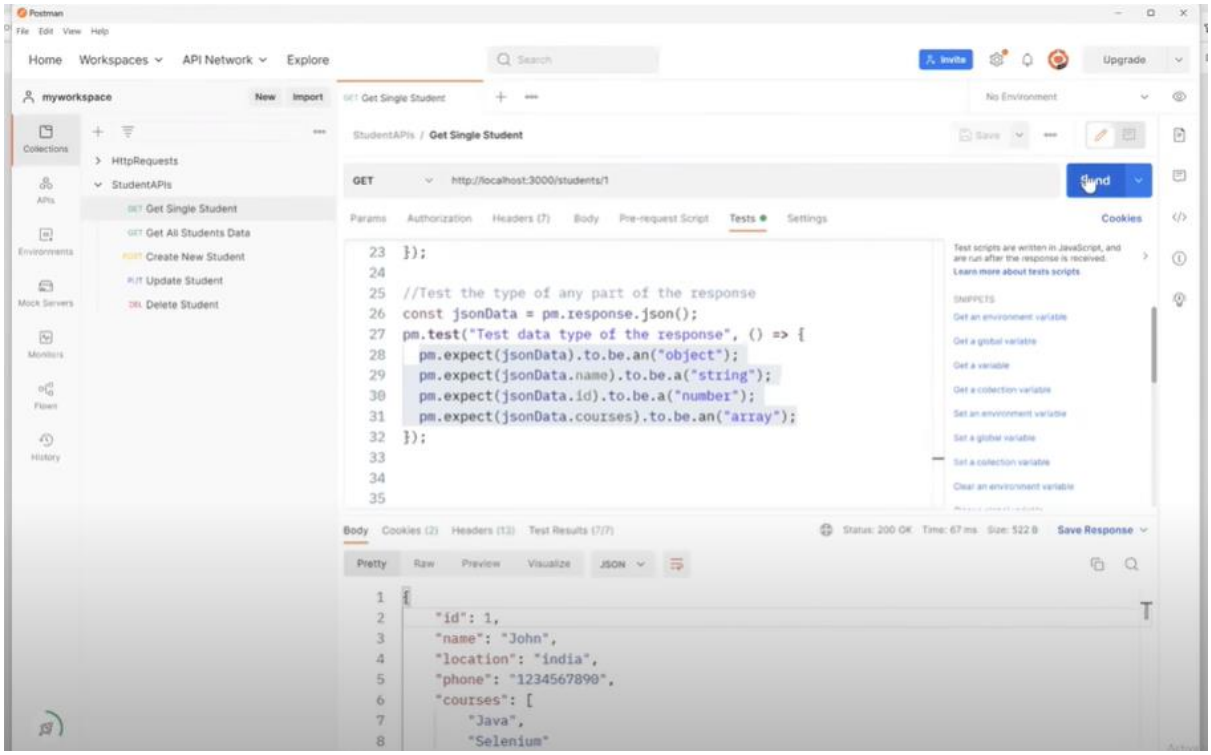

Home
Workspaces
API Network
Explore
myworkspace
New
Import
HttpRequests
StudentAPIs
Get Single Student
Get All Students Data
Create New Student
Update Student
Delete Student
StudentAPIs / Get Single Student
GET
http://localhost:3000/students/1
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
pm.expect(pm.response.headers.get('Content-Type')).to.eql
('application/json; charset=utf-8');
});
pm.test("Cookie 'language' is present", () => {
pm.expect(pm.cookies.has('language')).to.be.true;
});
pm.test("Cookie language has value 1", () => {
pm.expect(pm.cookies.get('language')).to.eql('en-gb');
});
Body
Cookies (2)
Headers (13)
Test Results (3/3)
Save Response
Name
Value
Domain
Path
Expires
HttpOnly
Secure
currency
USD
localhost
/
Wed, 10 Aug
false
false
language
en-gb
localhost
/
Wed, 10 Aug
false
false


Home
Workspaces
API Network
Explore
myworkspace
New
Import
HttpRequests
StudentAPIs
Get Single Student
Get All Students Data
Create New Student
Update Student
Delete Student
StudentAPIs / Get Single Student
GET
http://localhost:3000/students/1
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
pm.test("Cookie 'language' is present", () => {
pm.expect(pm.cookies.has('language')).to.be.true;
});
pm.test("Cookie language has value 1", () => {
pm.expect(pm.cookies.get('language')).to.eql('en-gb');
});
Body
Cookies (2)
Headers (13)
Test Results (4/5)
Save Response
All
Passed
Skipped
Failed
Status code is 200
Content-Type header is present
Content-Type header is application/json
Cookie 'language' is present
Socket Initialization
DNS Lookup
TCP Handshake
Transfer Start
Download

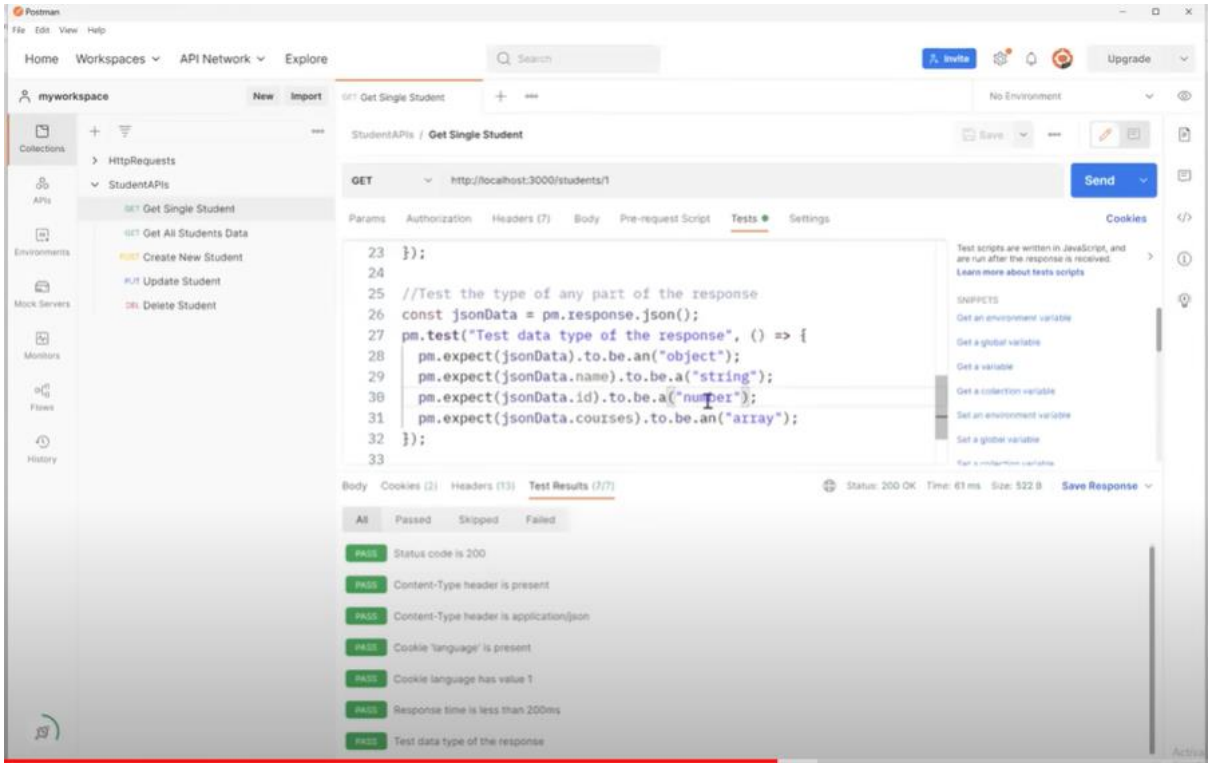


Response Body:






```
//Test the type of any part of the response
const jsonData = pm.response.json();
pm.test("Test data type of the response", () => {
   pm.expect(jsonData).to.be.an("object");
   pm.expect(jsonData.name).to.be.a("string");
   pm.expect(jsonData.id).to.be.a("number");
   pm.expect(jsonData.courses).to.be.an("array");
});
```

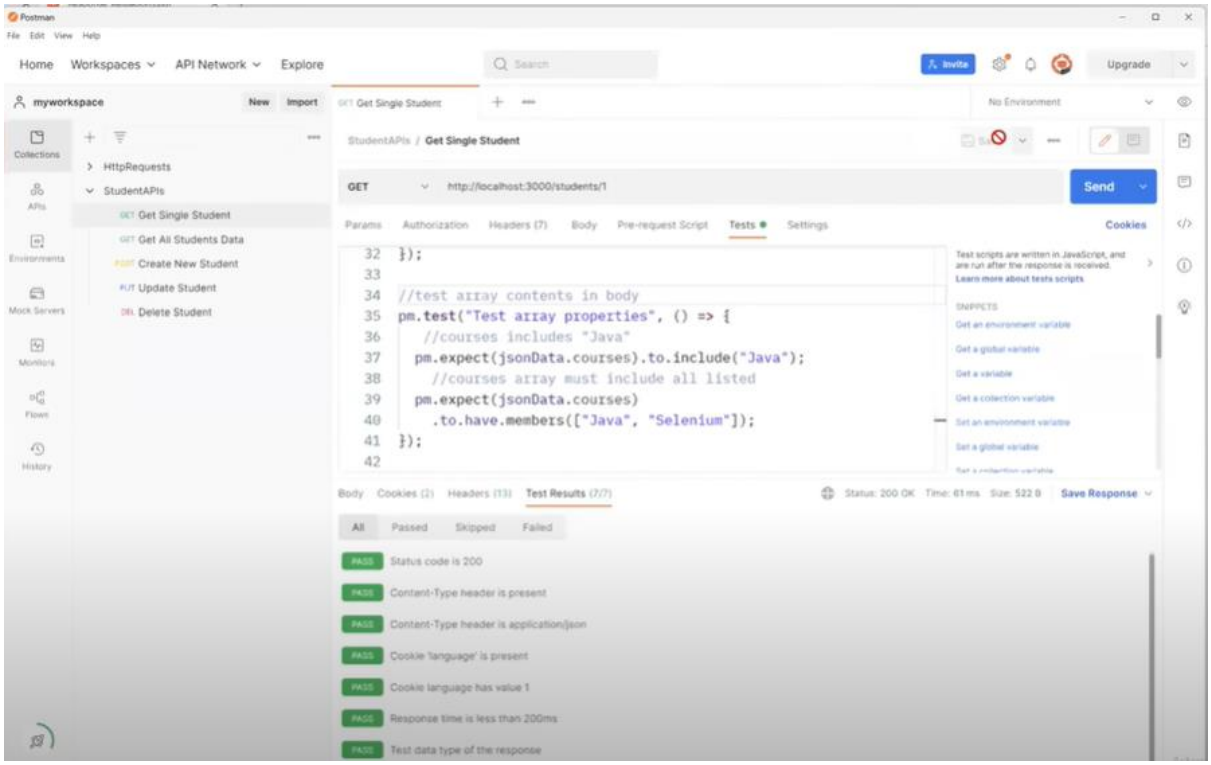


Asserting array properties

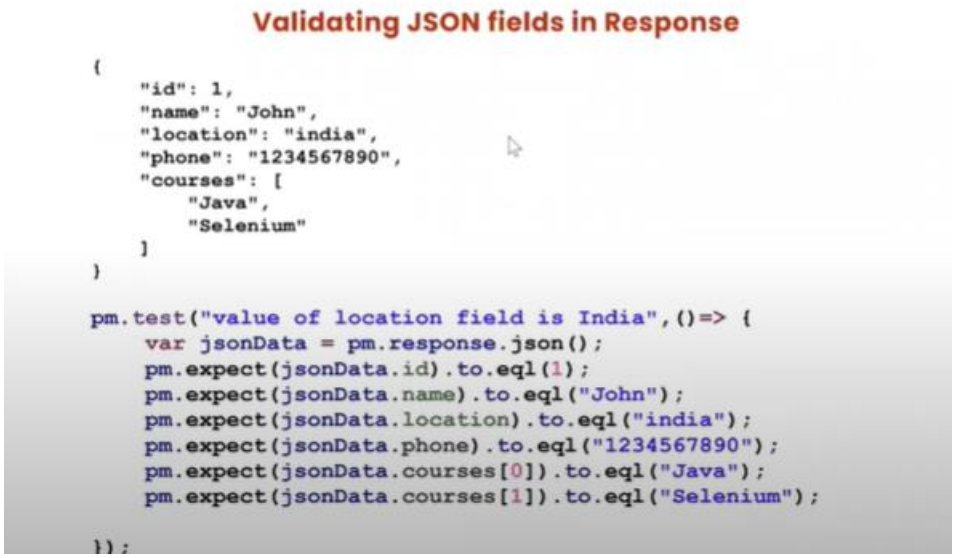
Check if an array is empty and if it contains particular items:

```
{
    "id": 1,
    "name": "John",
    "location": "india",
    "phone": "1234567890",
    "courses": [
        "Java",
        "Selenium"
    ]
}
```

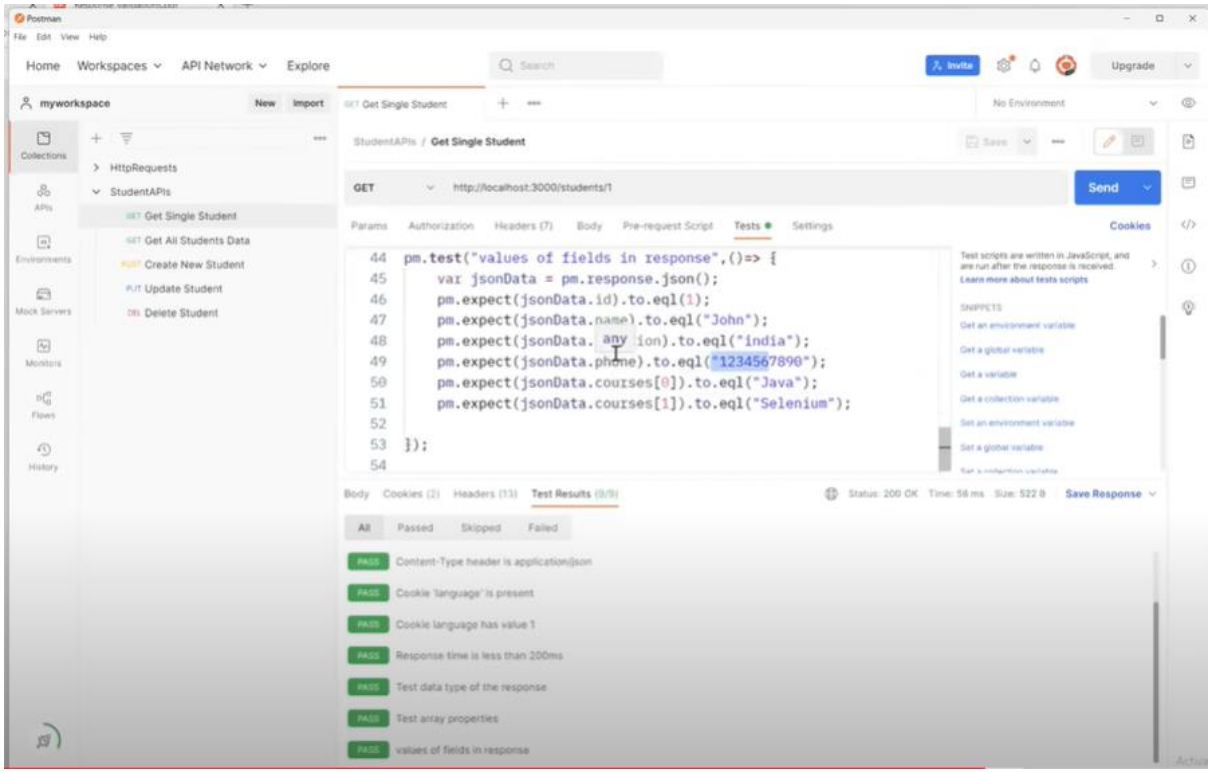
```
pm.test("Test array properties", () => {
   //courses includes "Java"
  pm.expect(jsonData.courses).to.include("Java");
    //courses array must include all listed
  pm.expect(jsonData.courses)
    .to.have.members(["Java", "Selenium"]);
});
```




```
//Test array contents in body
pm.test("Test array properties", () => {
   //course includer "Java"
   pm.expect(jsonData.courses).to.include("Java");
   //courses array must include all listed
   pm.expect(jsonData.courses).to.have.members(["Java", "Selenium"]);
});
```




```
//Validating JSON fields in Response
pm.test("value of fields in response", () => {
   var jsonData = pm.response.json();
   pm.expect(jsonData.id).to.eql(1);
   pm.expect(jsonData.name).to.eql("John");
   pm.expect(jsonData.location).to.eql("India");
   pm.expect(jsonData.phone).to.eql("1234567890");
   pm.expect(jsonData.courses[0]).to.eql("Java");
});
```



JSON Schema

Response

```
{
    "id": 1,
    "name": "John",
    "location": "india",
    "phone": "1234567890",
    "courses": [
        "Java",
        "Selenium"
    ]
}
```

JSON schema

```
var schema={
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "id": {
```

```
      "type": "integer"
    },
    "name": {
      "type": "string"
    },
    "location": {
      "type": "string"
    },
```

JSON Schema: https://www.liquid-technologies.com/online-json-to-schema-converter

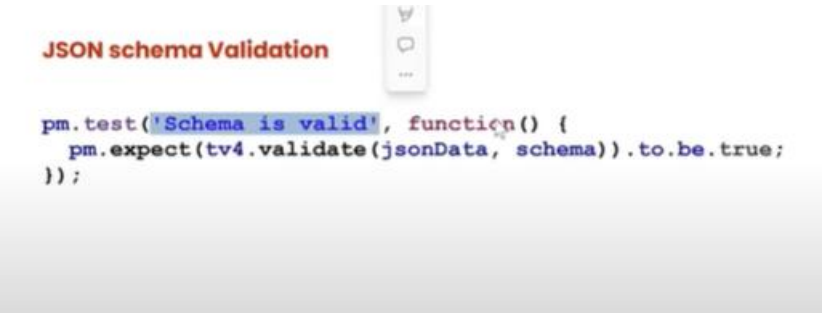
{
"id": 1,
"name": "John",
"location": "India",
"phone": "1234567890",
"courses": [
"Java",
"Selenium"
]
}

Generate Schema:

JSON Schema:

{
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"properties": {
"id": {
"type": "integer"
},
"name": {
"type": "string"
},
"location": {
"type": "string"
},
"phone": {
"type": "string"
},

```
    "courses": {
      "type": "array",
      "items": [
        {
          "type": "string"
        },
        {
          "type": "string"
        }
      ]
    }
  },
  "required": [
    "id",
    "name",
    "location",
    "phone",
    "courses"
  ]
}
```



Ex:

```
//JSON Schema Validation
var schema = {
```

```
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"properties": {
  "id": {
    "type": "string"
  },
  "name": {
    "type": "string"
  },
  "location": {
    "type": "string"
  },
  "phone": {
    "type": "string"
  },
  "courses": {
    "type": "array",
    "items": [
      {
        "type": "string"
      },
      {
        "type": "string"
      }
    ]
  }
},
"required": [
  "id",
```

```
    "name",
    "location",
    "phone",
    "courses"
  ]
}

pm.test('Schema is a valid', function() {
    pm.expect(tv4.validate(jsonData, schema)).to.be.true;
});
```

Example complete operations:

```
pm.test("Status code is 200", () => {
    pm.response.to.have.status(200);
});

pm.test("Successful GET resquest", () => {
    pm.expect(pm.response.code).to.be.oneOf([200,201]);
});

pm.test("Content-Type header is present", () => {
    pm.response.to.have.header("Content-Type");
});

pm.test("Content-Type header is application/json", () => {
    pm.expect(pm.response.headers.get("Content-Type")).to.eql('application/json');
    // pm.expect(pm.response.headers.get("Content-Type")).to.eql('application/json;
charset=utf-8');
});
```

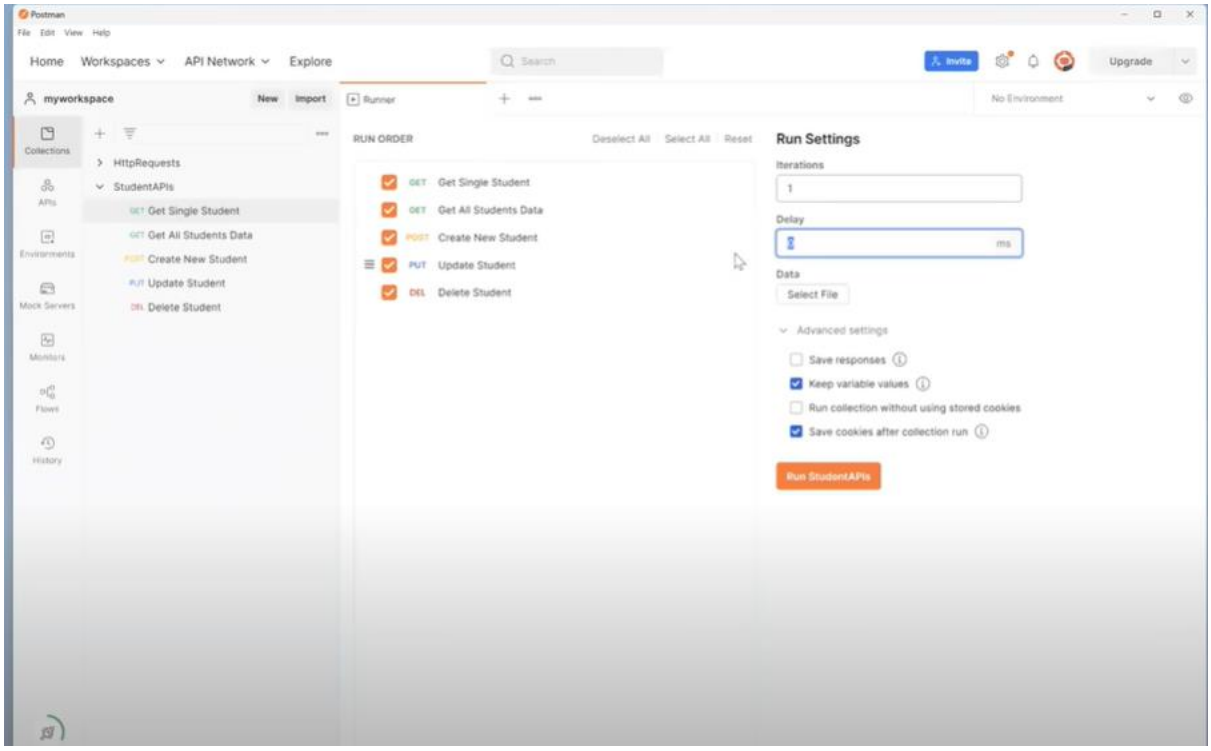
```
pm.test("Cookie 'language' is present", () => {
    pm.expect(pm.cookies.has('language')).to.be.true;
});

pm.test("Cookie language has value 1", () => {
    pm.expect(pm.cookies.get('language').to.eql('en-gb'));
});

pm.test("Response time is less than 50ms", () => {
    pm.expect(pm.response.responseTime).to.be.below(50);
});

//Test the type of any part of the response
const jsonData = pm.response.json();
pm.test("Test data type of the response", () => {
    pm.expect(jsonData).to.be.an("object");
    pm.expect(jsonData.name).to.be.a("string");
    pm.expect(jsonData.id).to.be.a("number");
    pm.expect(jsonData.courses).to.be.an("array");
});

//Test array contents in body
pm.test("Test array properties", () => {
    //course includer "Java"
    pm.expect(jsonData.courses).to.include("Java");
    //courses array must include all listed
    pm.expect(jsonData.courses).to.have.members(["Java", "Selenium"]);
});
```

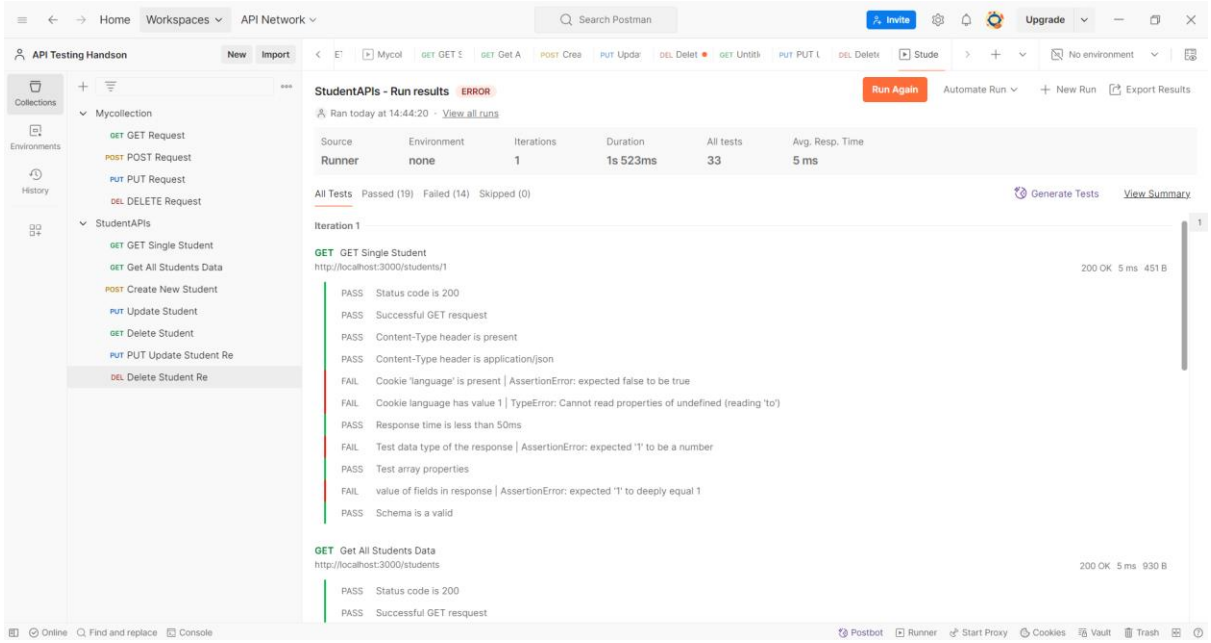
```
//Validating JSON fields in Response
pm.test("value of fields in response", () => {
  // var jsonData = pm.response.json();
   pm.expect(jsonData.id).to.eql(1);
   pm.expect(jsonData.name).to.eql("John");
   pm.expect(jsonData.location).to.eql("India");
   pm.expect(jsonData.phone).to.eql("1234567890");
   pm.expect(jsonData.courses[0]).to.eql("Java");
});

//JSON Schema Validation
var schema = {
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "id": {
      "type": "string"
    },
    "name": {
      "type": "string"
    },
    "location": {
      "type": "string"
    },
    "phone": {
      "type": "string"
    },
    "courses": {
      "type": "array",
```

```
      "items": [
        {
          "type": "string"
        },
        {
          "type": "string"
        }
      ]
    }
  },
  "required": [
    "id",
    "name",
    "location",
    "phone",
    "courses"
  ]
}
```

```
pm.test('Schema is a valid', function() {
    pm.expect(tv4.validate(jsonData, schema)).to.be.true;
});
```

Run Collection:





Scripts and Types of variables

Scripts

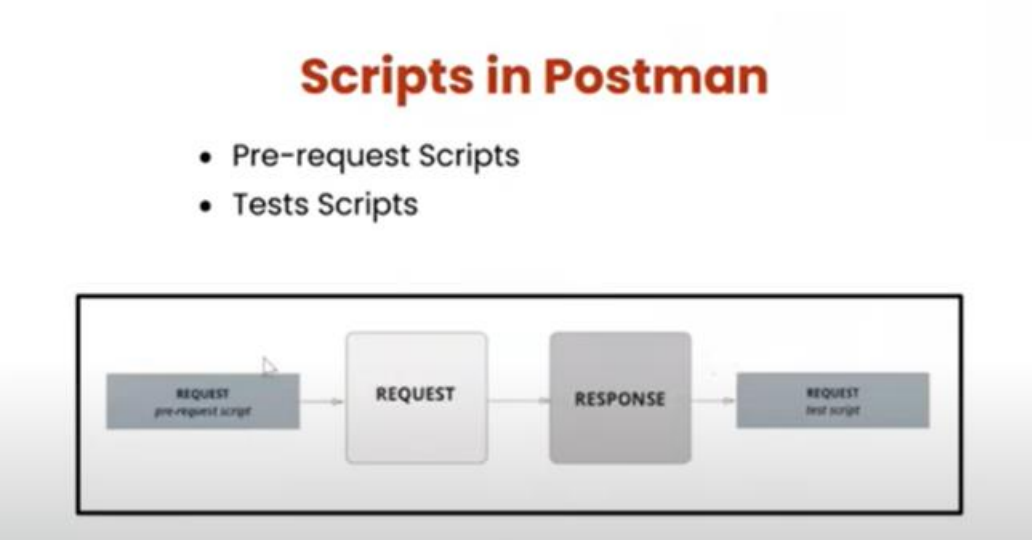
Pre-request scripts

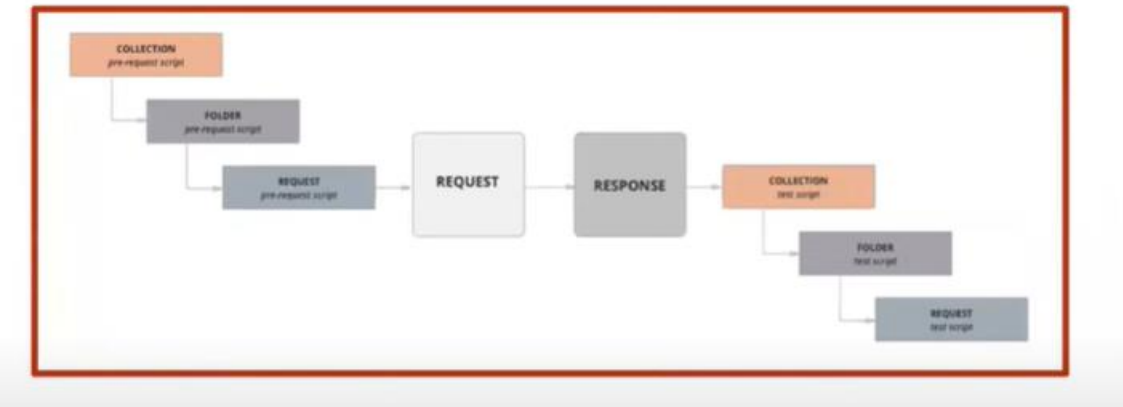
Tests

Pre-requestScript --→ Request --→ Response --→ Tests

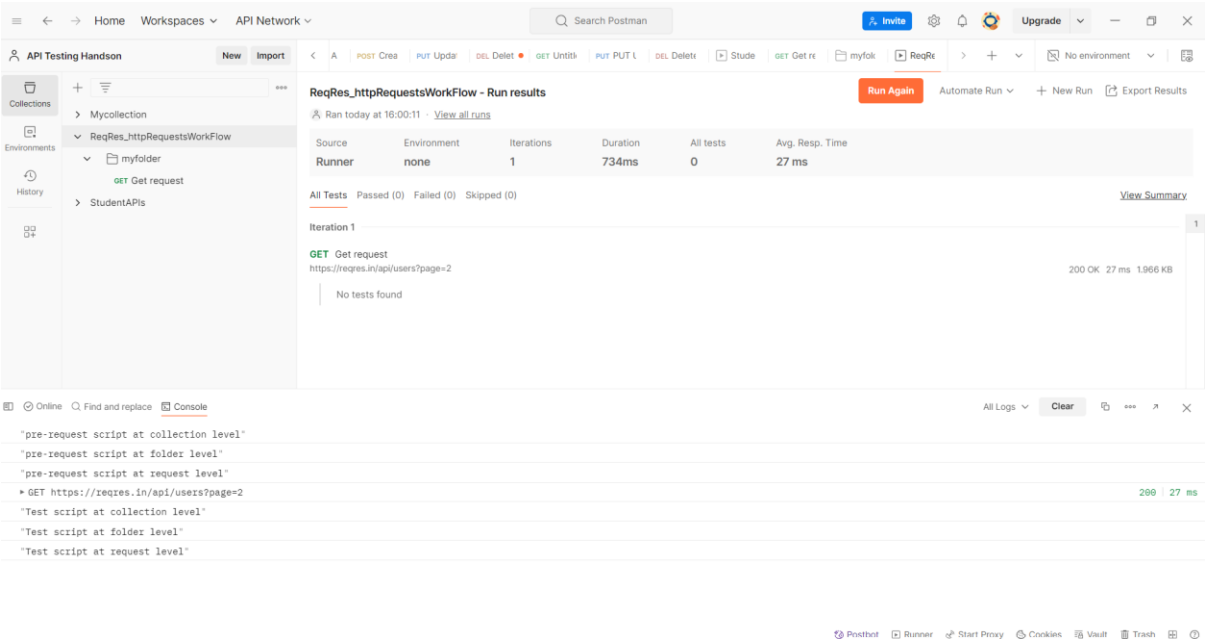
3 levels:

1) Collection
2) Folder
3) Request





Request response order:



Variables:

Scope:

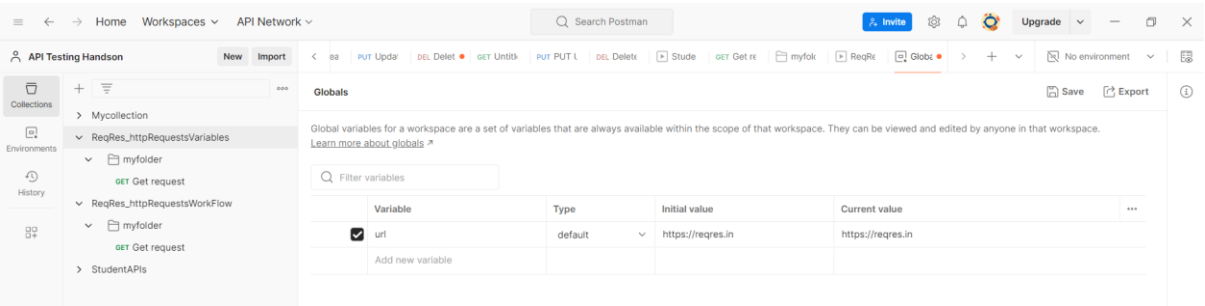
Workspace -→ Collections -→ Requests

Types of variables as per Scope of access of variables:

1) Global => accessible in workspace.
2) Collection => accessible within collection.
3) Environment => accessible in all collections, but we need to switch to environment.
4) Local => accessible only within request(specific to request). Declared inside Pre-request Script i.e. Request -> Scripts -> Pre-request.
5) Data => external files csv/text.

Referring variable: {{variable}}

Global variables:



Home Workspaces API Network Search Postman Invite Upgrade

API Testing Handson New Import

Collections
Environments
History

- Mycollection
  - GET GET Request
  - POST POST Request
  - PUT PUT Request
  - DEL DELETE Request
- ReqRes_httpRequestsVariables
  - myfolder
    - GET Get request
    - POST Post request
    - PUT PUT Request
    - DEL Delete request
- ReqRes_httpRequestsWorkFlow
  - myfolder
    - GET Get request
- StudentAPIs

DEL Delete | Stude | GET Get re | myfok | ReqRe | Globa | GET Get re | POST Post | PUT PUT F | DEL Delete | No environment

Globals Save Export

Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be viewed and edited by anyone in that workspace.
Learn more about globals

Filter variables

Variable	Type	Initial value	Current value
☑ url	default	https://reqres.in	https://reqres.in
Add new variable			

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

Home Workspaces API Network Search Postman Invite Upgrade

API Testing Handson New Import

Collections
Environments
History

- Mycollection
  - GET GET Request
  - POST POST Request
  - PUT PUT Request
  - DEL DELETE Request
- ReqRes_httpRequestsVariables
  - myfolder
    - GET Get request
    - POST Post request
    - PUT PUT Request
    - DEL Delete request
- ReqRes_httpRequestsWorkFlow
  - myfolder
    - GET Get request
- StudentAPIs

DEL Delete | Stude | GET Get re | myfok | ReqRe | Globa | GET Get re | POST Post | PUT PUT F | DEL Delete | No environment

ReqRes_httpRequestsVariables / myfolder / **Get request** Save Share

GET {{url}}/api/users?page=2 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

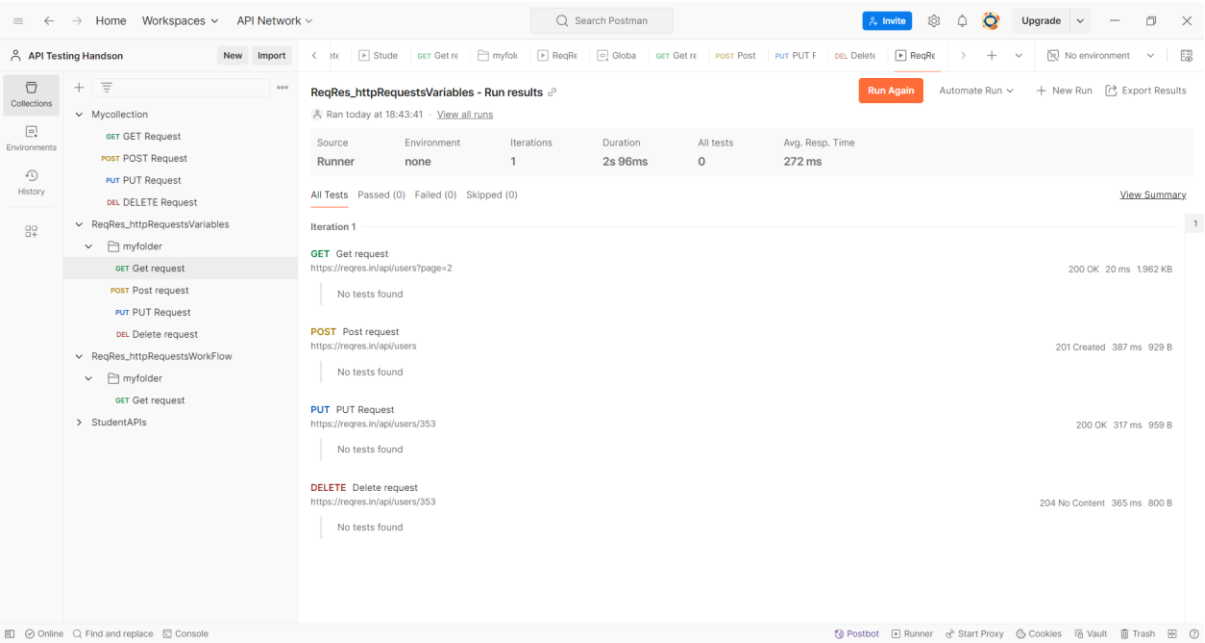
Key	Value	Description
☑ page	2	
Key	Value	Description

Body Cookies Headers (18) Test Results Status: 200 OK Time: 27 ms Size: 1.92 KB Save as example

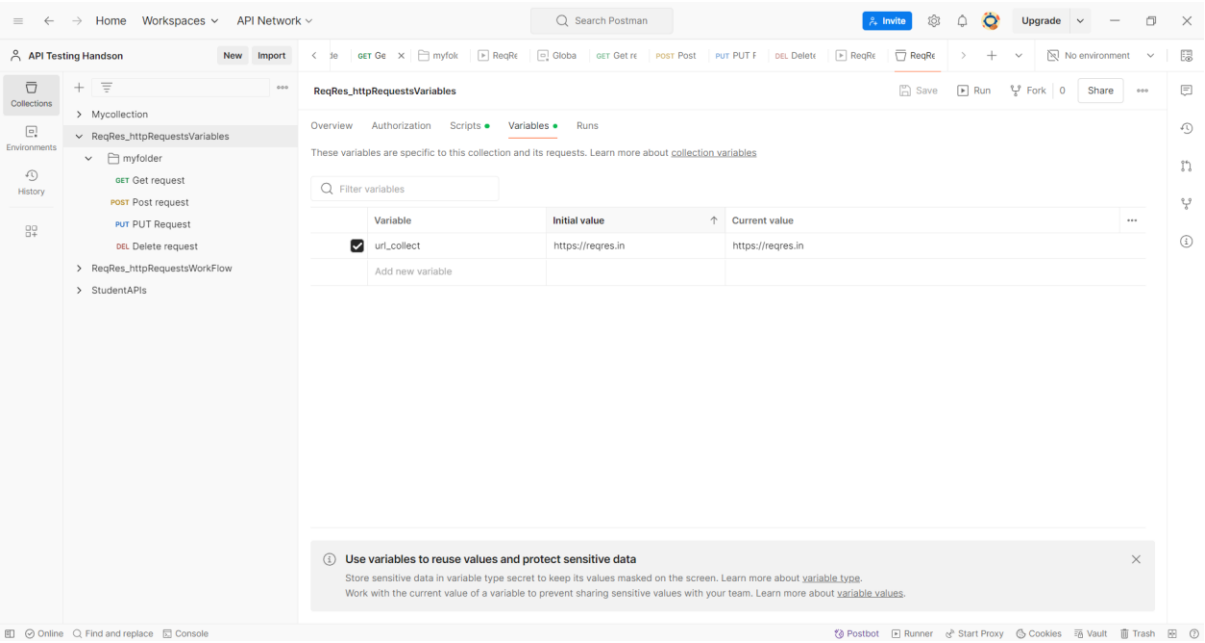
Pretty Raw Preview Visualize JSON

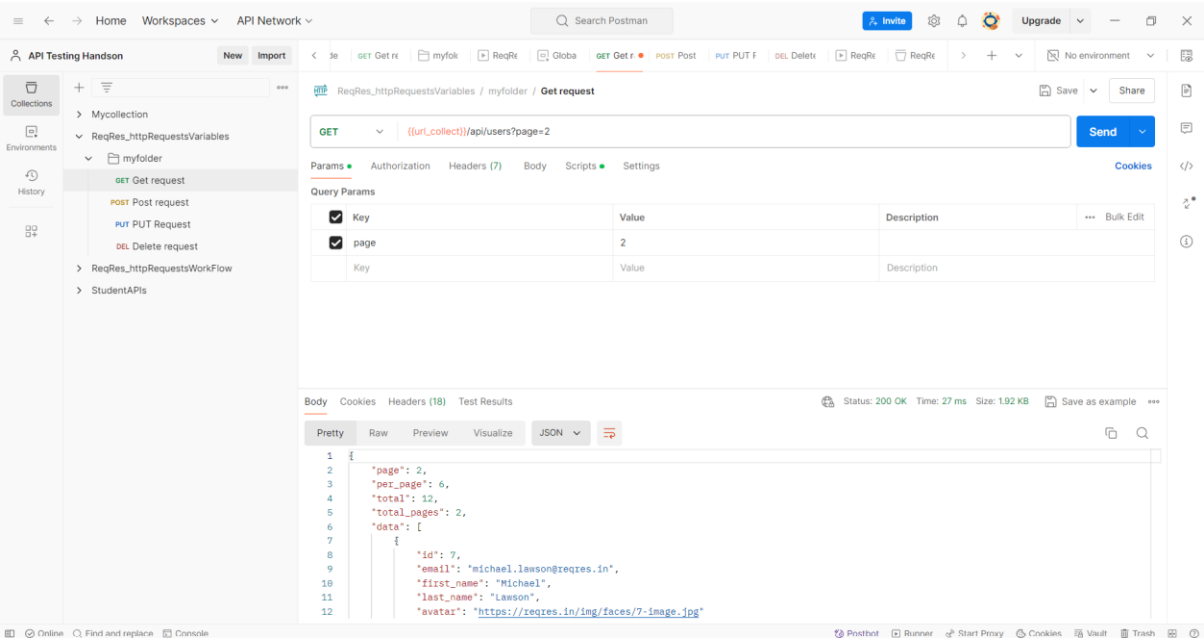

```
{
    "page": 2,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [
        {
            "id": 7,
            "email": "michael.lawson@reqres.in",
            "first_name": "Michael",
            "last_name": "Lawson",
            "avatar": "https://reqres.in/img/faces/7-image.jpg"
```


Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

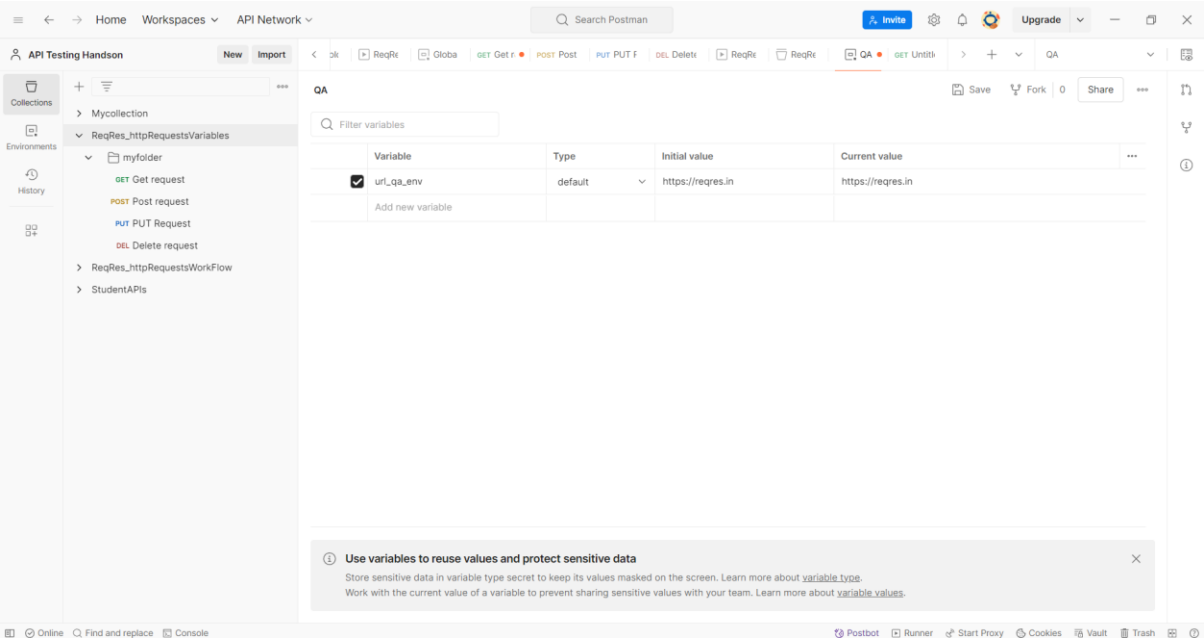


Collection variable:

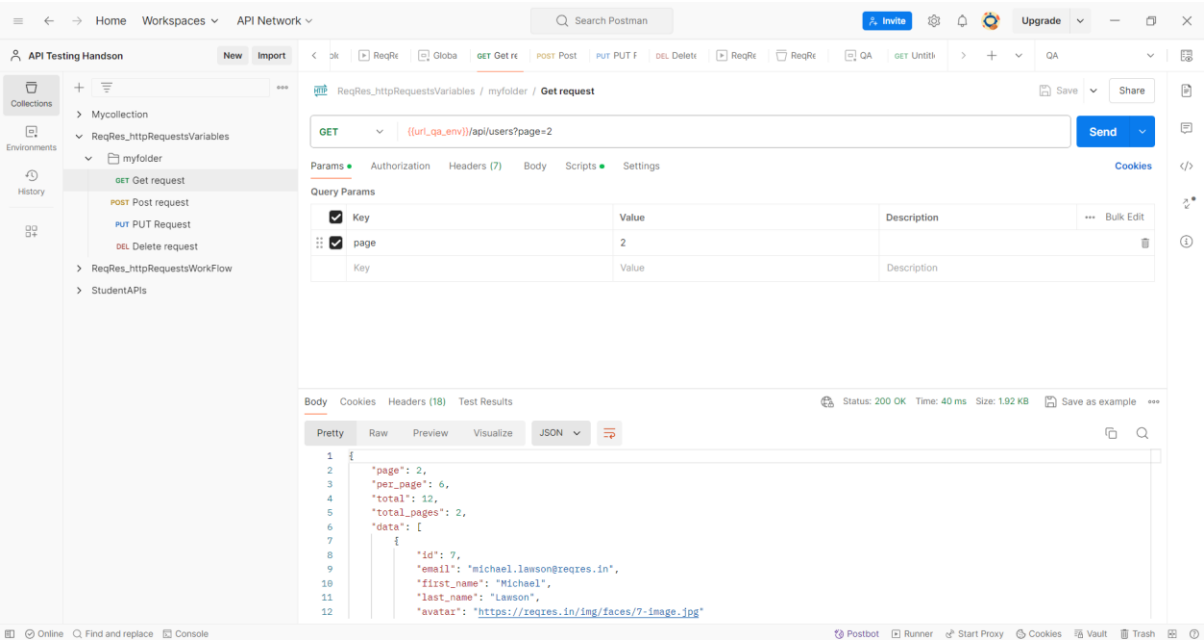




Environment variables:



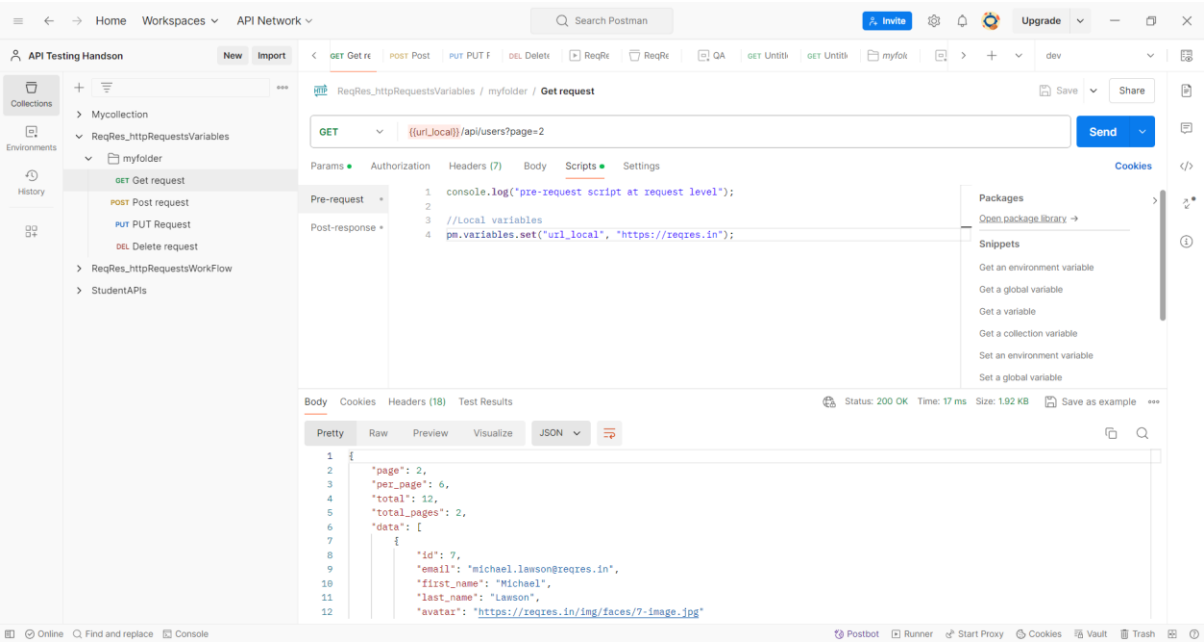
Select environment as QA.



Local Variable:

//Local variables

pm.variables.**set**("url_local", "https://reqres.in");



We can also initialize global variable under request

Ex:

GET => {{url_local}}/api/users?page={{userid_globals}}

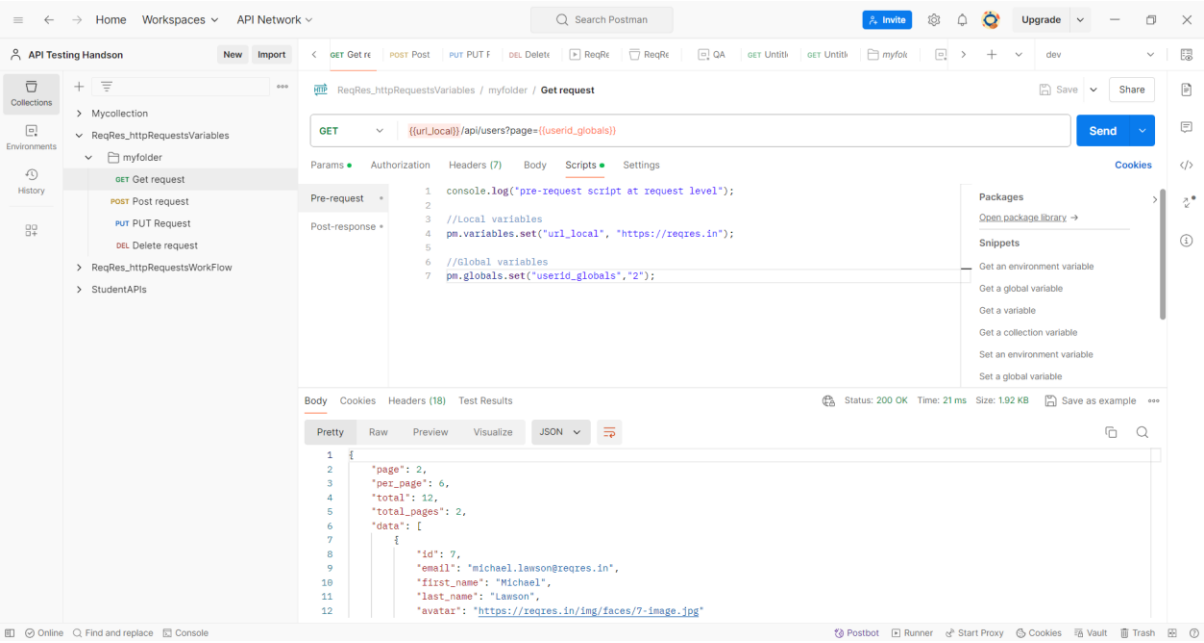
Script=>

//Local variables

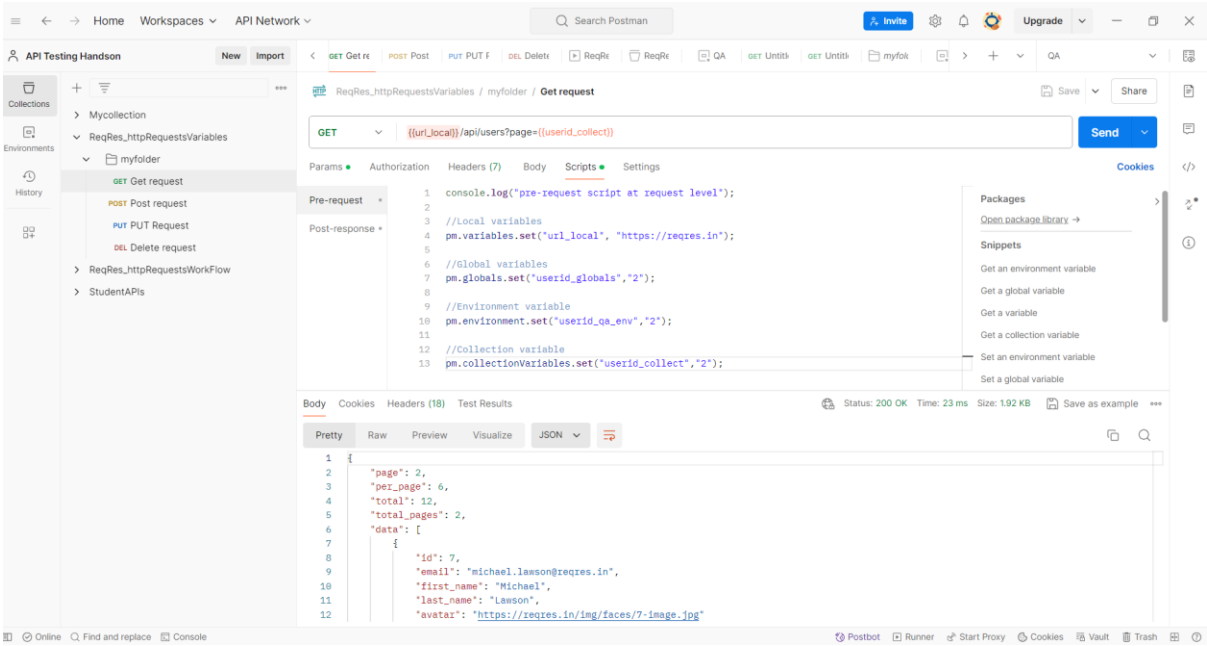
pm.variables.**set**("url_local", "https://reqres.in");

//Global variables

pm.globals.**set**("userid_globals","2");



A global variable created in a request can also be accessible everywhere.



//Local variables

pm.variables.**set**("url_local", "https://reqres.in");

//Global variables

```
pm.globals.set("userid_globals","2");

//Environment variable
pm.environment.set("userid_qa_env","2");

//Collection variable
pm.collectionVariables.set("userid_collect","2");
```

Creating variables using pre-request scripts:

```
//Local variables
pm.variables.set("url_local", "https://reqres.in");

//Global variables
pm.globals.set("userid_globals","2");

//Environment variable
pm.environment.set("userid_qa_env","2");

//Collection variable
pm.collectionVariables.set("userid_collect","2");
```

Unset/Delete: in Post-response Scripts

Ex:

```
//Global variables
pm.globals.unset("userid_globals");
```

Capture the values from variables:

```
console.log(pm.globals.get("userid_globals"));
console.log(pm.environment.get("userid_qa_env"));
console.log(pm.collectionVariables.get("userid_collect"));
console.log(pm.variables.get("url_local"));
```

Methods:

Set, unset, get => global, env, collection, local.

API Chaining

Ex: https://gorest.co.in/

GET: https://gorest.co.in/public/v2/users

GoRest API :

https://gorest.co.in/

Generate Token using github account

url: https://gorest.co.in/

endpoint: /public/v2/users

REST API Http Response Codes

- 200: OK. Everything worked as expected.
- 201: A resource was successfully created in response to a POST request. The Location header contains the URL pointing to the newly created resource.
- 204: The request was handled successfully and the response contains no body content (like a DELETE request).
- 304: The resource was not modified. You can use the cached version.
- 400: Bad request. This could be caused by various actions by the user, such as providing invalid JSON data in the request body etc.
- 401: Authentication failed.
- 403: The authenticated user is not allowed to access the specified API endpoint.
- 404: The requested resource does not exist.
- 405: Method not allowed. Please check the Allow header for the allowed HTTP methods.
- 415: Unsupported media type. The requested content type or version number is invalid.
- 422: Data validation failed (in response to a POST request, for example). Please check the response body for detailed error messages.
- 429: Too many requests. The request was rejected due to rate limiting.
- 500: Internal server error. This could be caused by internal program errors.

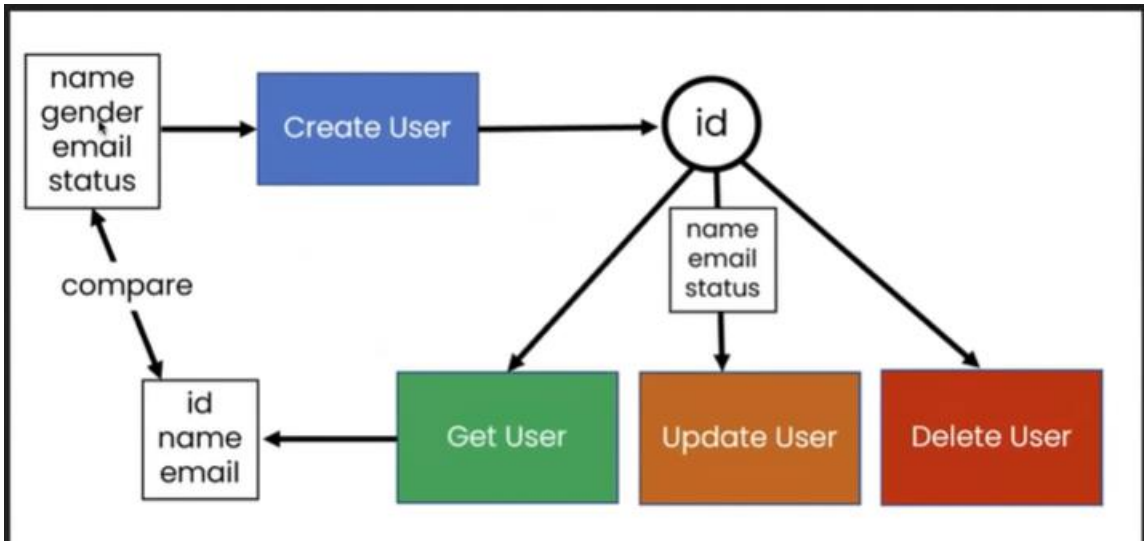
Request Body:

```
{
   "name": "rutuja",
   "email": "test@gmail.com",
```

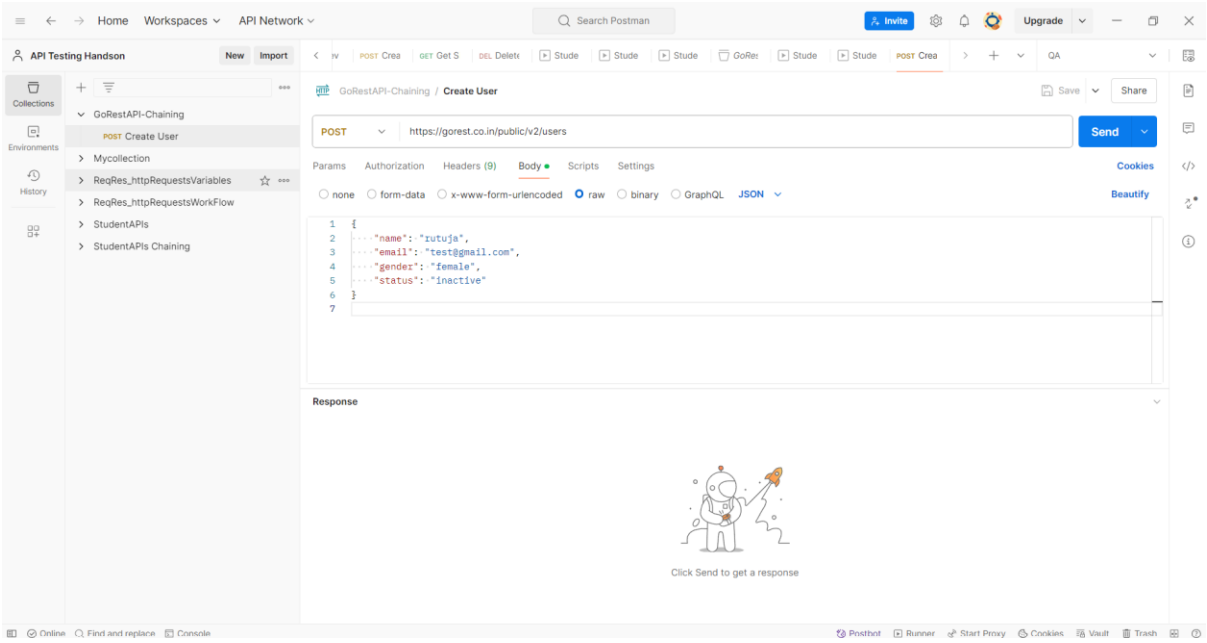
"gender": "female",

"status": "inactive"

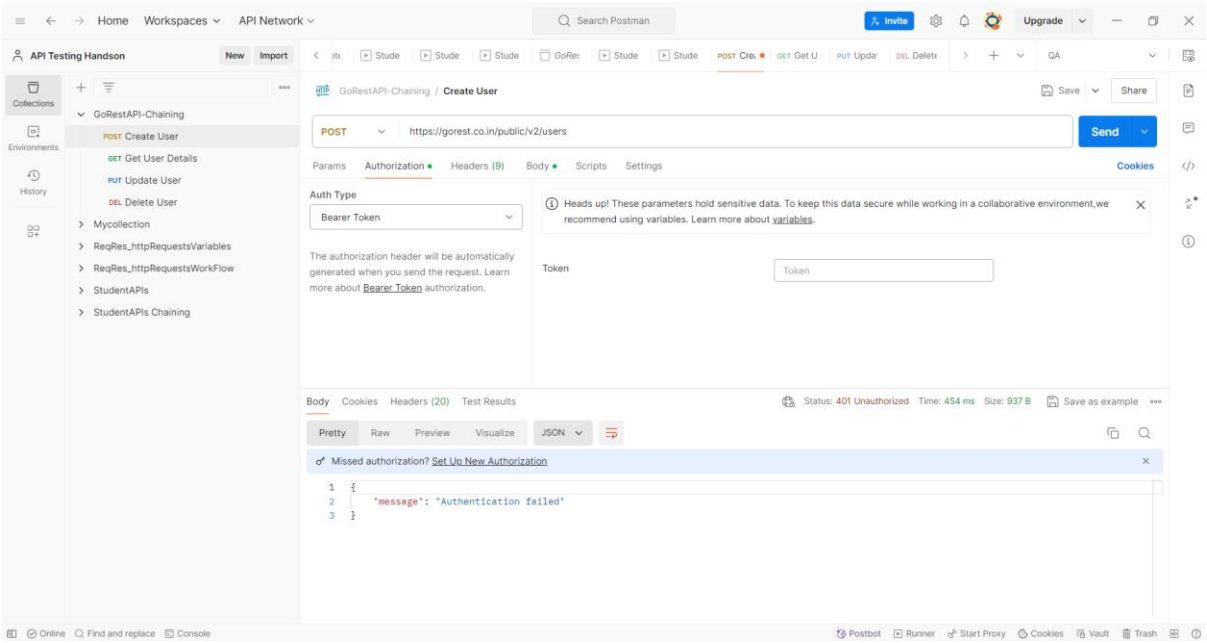
}



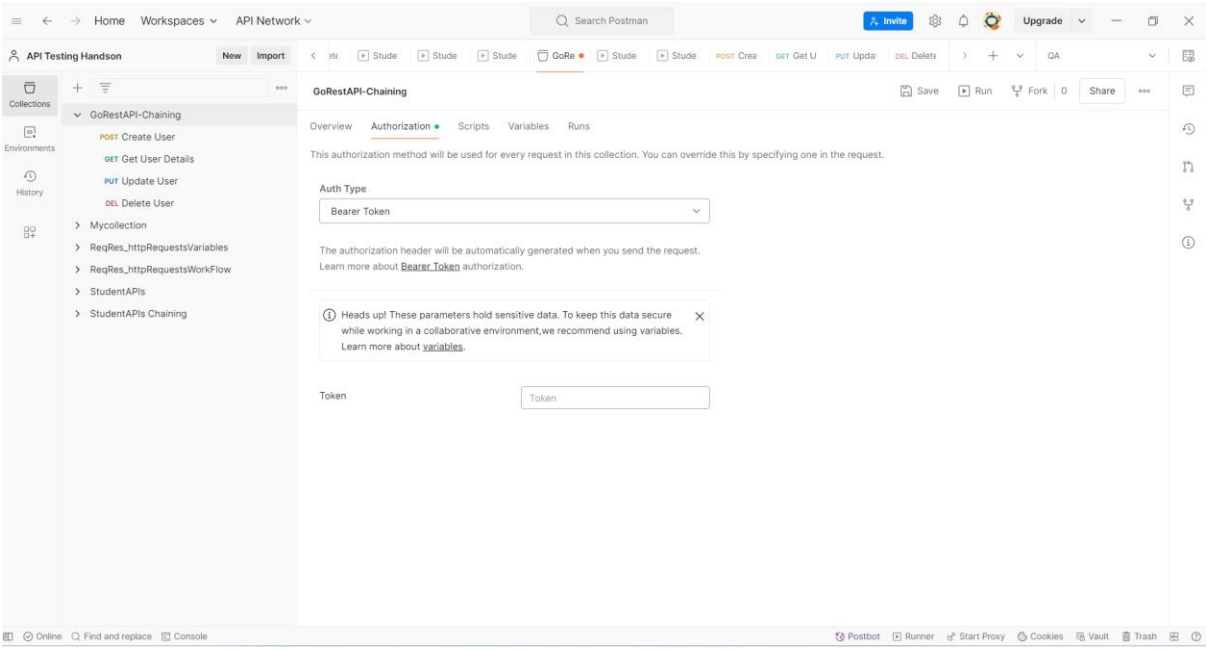
POST:



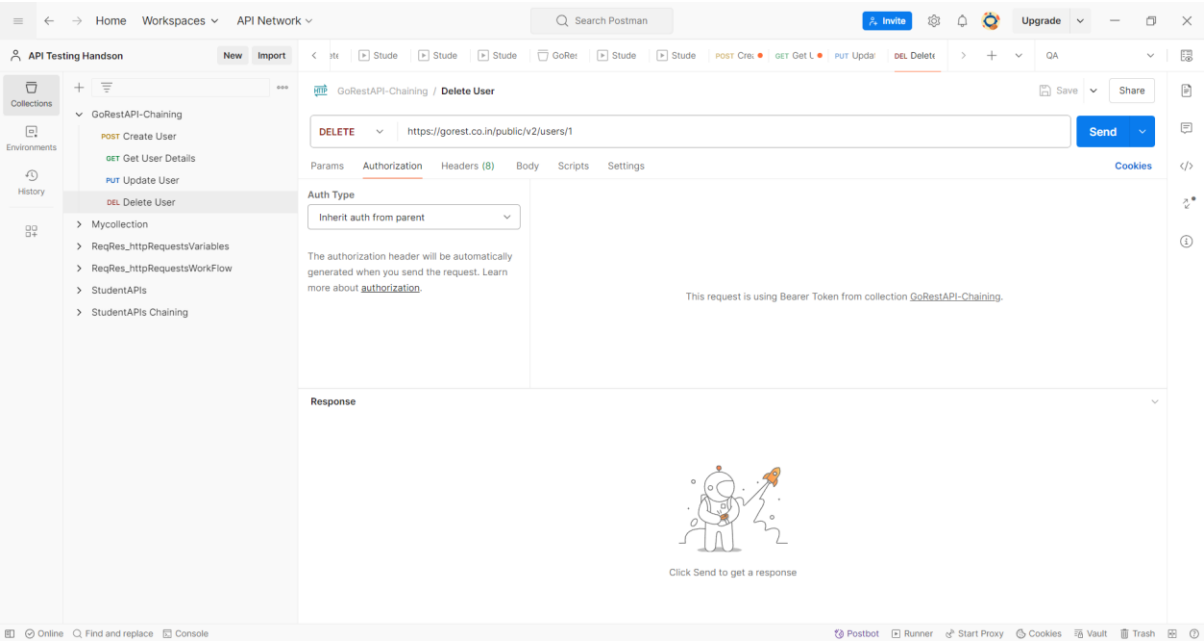
Provide Bearer Token Value.



Add Authorization i.e. Bearer Token at collection level to avoid repetitive task.



Now for Request -> under Authorization -> select Inherit Auth from Parent.


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
API Testing Handson
New
Import
Collections
Environments
History
GoRestAPI-Chaining
POST Create User
GET Get User Details
PUT Update User
DEL Delete User
Mycollection
ReqRes_httpRequestsVariables
ReqRes_httpRequestsWorkFlow
StudentAPIs
StudentAPIs Chaining
GoRestAPI-Chaining / Delete User
Save
Share
DELETE
https://gorest.co.in/public/v2/users/1
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Auth Type
Inherit auth from parent
The authorization header will be automatically generated when you send the request. Learn more about authorization.
This request is using Bearer Token from collection GoRestAPI-Chaining.
Response
Click Send to get a response
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

abcd