POSTMAN

(a basic guide for REST API Testing)

created Akash Jindal

ACKNOWLEDGEMENT

Document is been written after the much-required knowledgeable sessions received from

DEEPAK CHANANA

Many Thanks DEEPAK CHANANA for the best of your support, motivation and knowledge you always try to give to your students at very best.

Thank you for taking out your valuable time out of your schedule and bringing out the best of many of us.

Thank you for enhancing the knowledge, boosting up the confidence and shaping the career path of many of us with your continuous support as mentor, as friend, as brother every time and at any time.

Application Programming Interface (API) :

Is the way of communication between two applications where applications may differ in their platforms or in terms of technology also.

Features and Resources :

'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.

Types Of API :

There are two types of API Automation, Simple Object Access Protocol (SOAP) and REST Assured (Representational State Transfer). Both are the web services.

URI - Uniform Resource Indicator :

Uniform resource indicator is a link at which one must perform API testing. It is provided by the developer. It consists of two parts- the Base and the endpoint.

Precisely look at the URI: - "http://localhost:3000/friends"

http://localhost:3000 **-Base**	/friends **-End-point**
(Location where API is deployed)	(Generally resource- functionality)

Example :

http://localhost:3000/friends. Here, "/friends" is the resource and rest of the portion "http://localhost:3000" is the part of Base and is the location where the API is deployed.

HTTP Methods :

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.

Below are the various http requests or methods we use in API automation testing

Http Request	Description
1. **Get**	No data is required for this request. it is used to fetch the data from thegiven resource location.
2. **Post**	Data is required for this kind of request. It is used to insert the data at thegiven resource location.
3. **Put**	Data is required for this request. Whole of the data is passed that is to be updated and the one that is not to be updated. Requires certain ID value too to update that particular record.
4. **Patch**	Data is required to for this http request also. Only that data is passed whichis required to be updated, requires certain ID value too to update that record
5. **Delete**	No data is required to this request. It deletes the record (data) from the given resource location. Requires certain ID value too to update thatrecord.

HTTP Response and Status Code :

HTTP Response is the server's information as a result of the client's request.

Status Codes are issued by a server in response to a client's request made to the server.

(Browse the https://en.wikipedia.org/wiki/List_of_HTTP_status_codes URL for list of HTTP Response Status Codes)

JavaScript Object Notation (JSON) :

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. Just remember two things now that Json have data in the form of object(s) and Array(s).

JSON is a data format/Notation that is used by the web services to transmit over the communication channel for an instance JSON is often used when data is sent from a server to a web page. JSON is "self- describing" and easy to understand.

JSON Example :

This example defines a Student object: an array of 3 employee records (objects) :

```
{
"Student":[
{"firstName":"Johny", "lastName":"Doe"},
{"firstName":"Ananiya", "lastName":"Smithy"},
{"firstName":"Raghubir", "lastName":"Singh"}]
}
```

JSON Syntax Rules :

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

The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects. The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

JSON data (Simple JSON) :

JSON data is written as name/value pairs, just like JavaScript object properties. A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value :

```
{"firstName":"Johny"}
```

JSON names require double quotes. JavaScript names do not.

JSON Objects (Complex JSON) :

JSON objects are written inside curly braces. Just like in JavaScript, objects can contain multiple name/value pairs :

```
“Personal”:
{
{"firstName":"Johny",
"lastName":"Doe"
},
“Id”:”369”,
“Age”:”38”
}
```

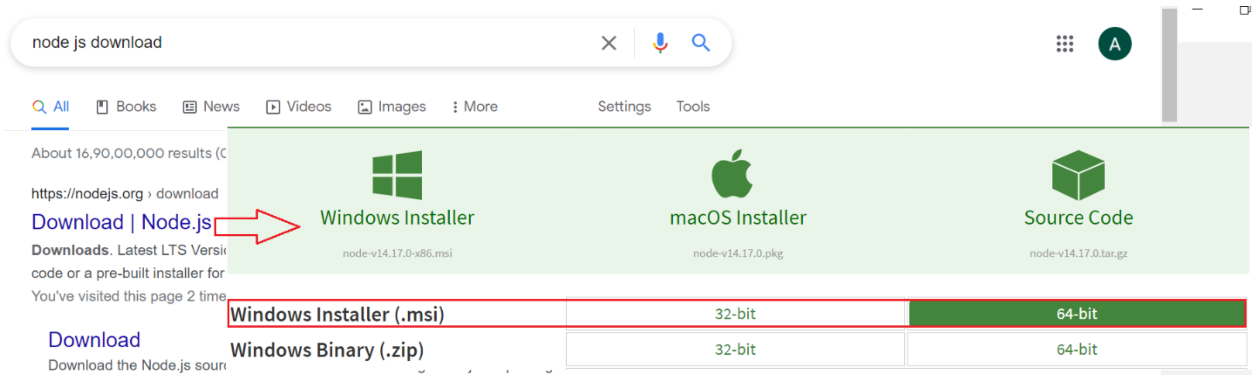
JSON Arrays (JSON in the Form of Array) :

JSON arrays are written inside square brackets. Just like in JavaScript, an array can contain objects :

```
"Student": [
{"firstName":"Johny", "lastName":"Doe"},
{"firstName":"Ananiya", "lastName":"Smithy"}   ]
```

Steps To Create Dummy API's For Testing

1. Install **node js** for creating dummy API's. Open a browser and download node js.



node js download

All | Books | News | Videos | Images | More | Settings | Tools

About 16,90,00,000 results (C

https://nodejs.org › download

Download | Node.js

Downloads. Latest LTS Versi
code or a pre-built installer for
You've visited this page 2 time

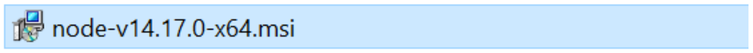
Download

Download the Node.js sour

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit

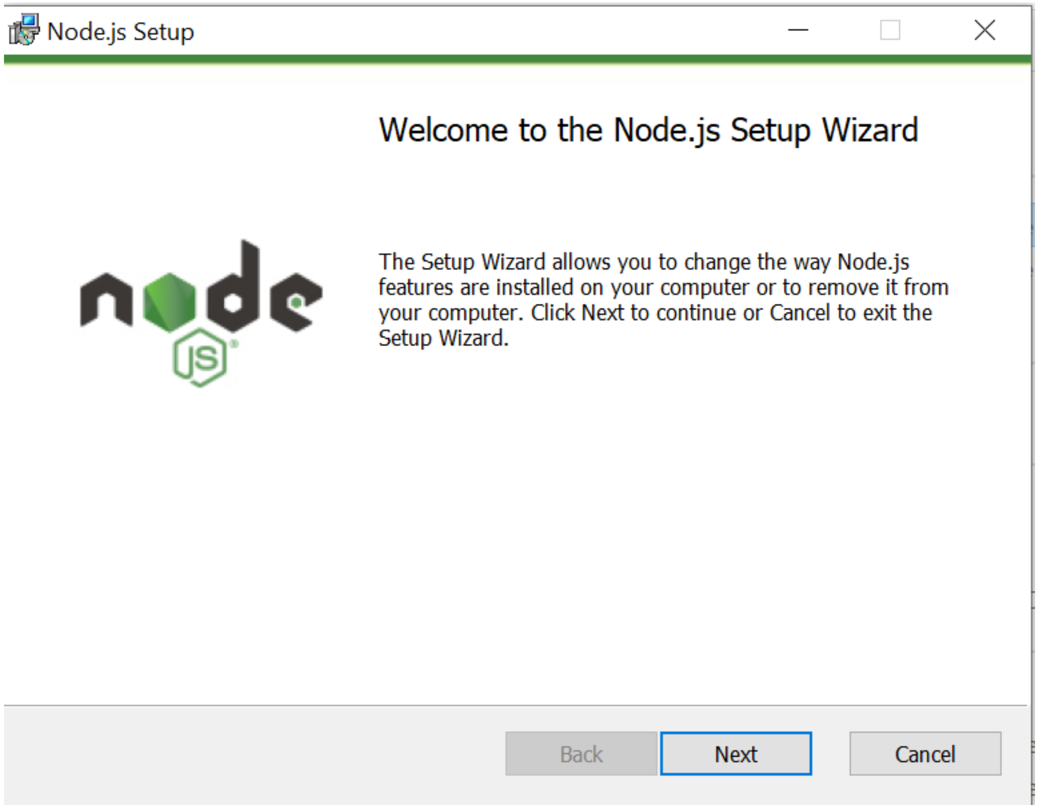
2. Download the setup meeting your system requirement.

3. Downloaded file will be an executable file :



node-v14.17.0-x64.msi

4. Simply click on the setup and install.



Welcome to the Node.js Setup Wizard

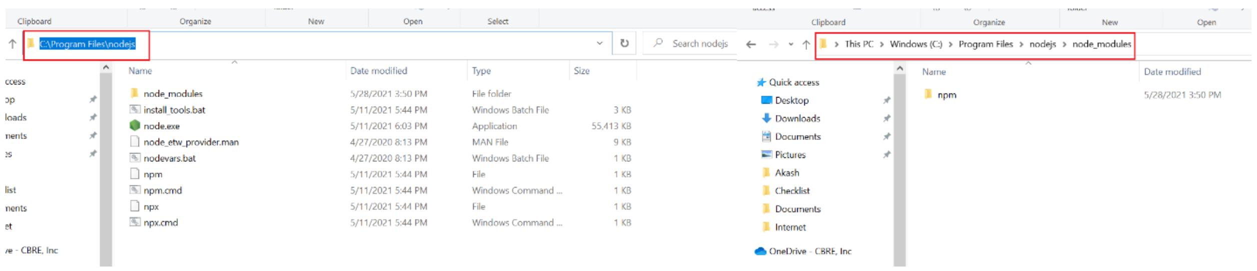
The Setup Wizard allows you to change the way Node.js features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

This will enable the system with node js setup.

5. Installing the node js setup provide us with the file named : **npm**.

6. **npm : Node Package Manager :** helps us in installing any of the node js package.

7. Copy the path of the **node js** setup and **node_modules** directory :



C:\Program Files\nodejs

Name	Date modified	Type	Size
node_modules	5/28/2021 3:50 PM	File folder	
install_tools.bat	5/11/2021 5:44 PM	Windows Batch File	3 KB
node.exe	5/11/2021 6:03 PM	Application	55,413 KB
node_etw_provider.man	4/27/2020 8:13 PM	MAN File	9 KB
nodevars.bat	4/27/2020 8:13 PM	Windows Batch File	1 KB
npm	5/11/2021 5:44 PM	File	1 KB
npm.cmd	5/11/2021 5:44 PM	Windows Command ...	1 KB
npx	5/11/2021 5:44 PM	File	1 KB
npx.cmd	5/11/2021 5:44 PM	Windows Command ...	1 KB

This PC › Windows (C:) › Program Files › nodejs › node_modules

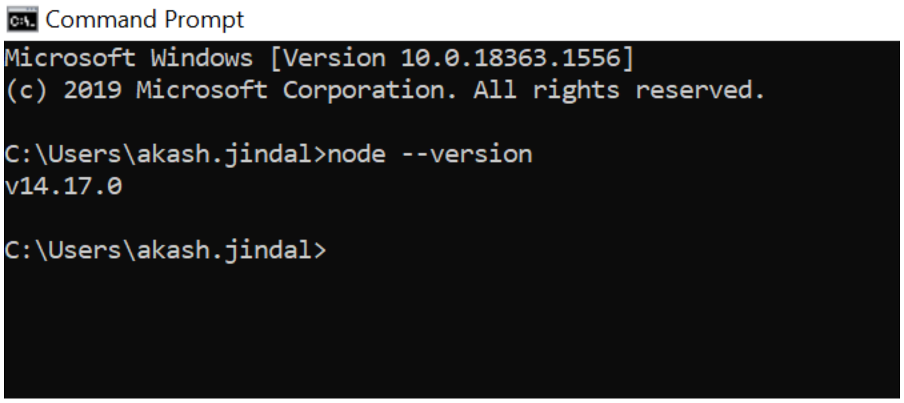
Name	Date modified
npm	5/28/2021 3:50 PM

and make an entry of the same under :

System Properties → Environment Variables → System Variables → Path

8. Verify the installed version of the node js under cmd :

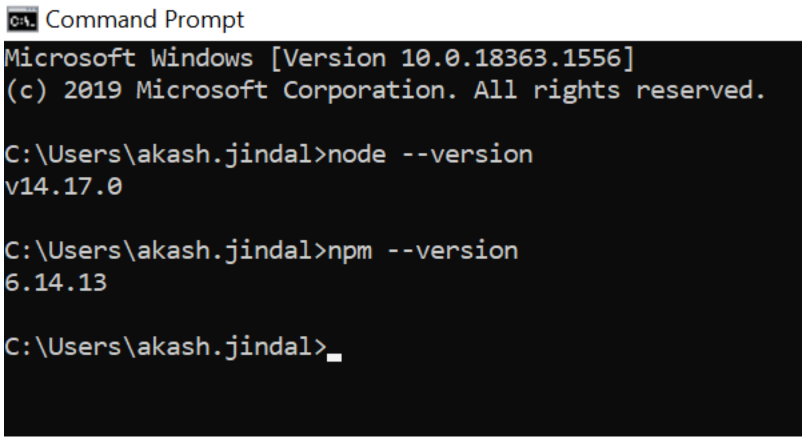
command : node --version


Command Prompt

```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\akash.jindal>node --version
v14.17.0

C:\Users\akash.jindal>
```

9. Verify the installed version of the npm under cmd :

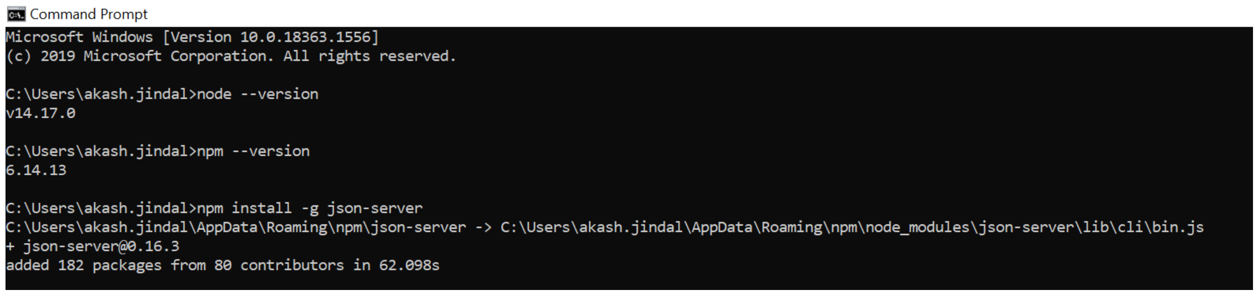
command : npm --version


Command Prompt

```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\akash.jindal>node --version
v14.17.0

C:\Users\akash.jindal>npm --version
6.14.13

C:\Users\akash.jindal>
```

10. Install the **json-server package** with the help of npm :

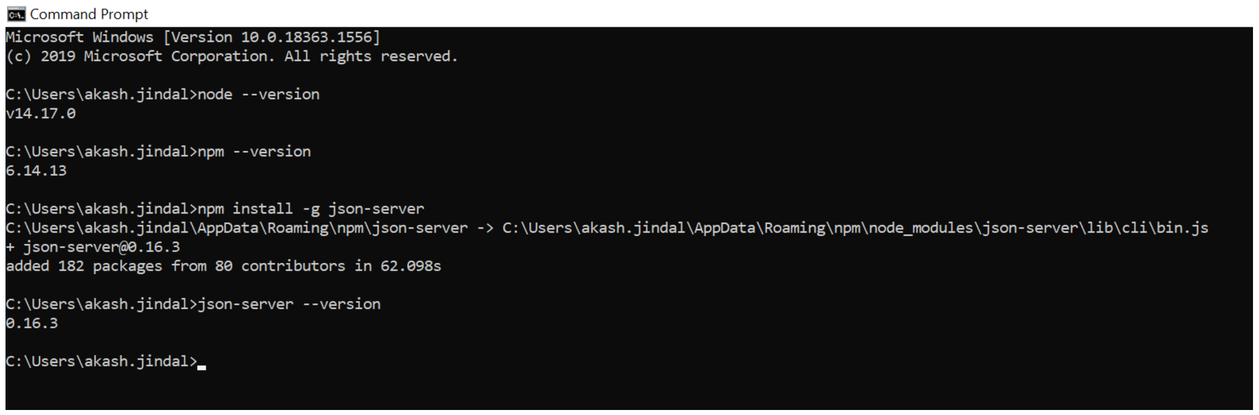
command : npm install -g json-server



```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\akash.jindal>node --version
v14.17.0

C:\Users\akash.jindal>npm --version
6.14.13

C:\Users\akash.jindal>npm install -g json-server
C:\Users\akash.jindal\AppData\Roaming\npm\json-server -> C:\Users\akash.jindal\AppData\Roaming\npm\node_modules\json-server\lib\cli\bin.js
+ json-server@0.16.3
added 182 packages from 80 contributors in 62.098s
```

11. Verify the installed version of the json-server under cmd :

command : json-server --version



```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\akash.jindal>node --version
v14.17.0

C:\Users\akash.jindal>npm --version
6.14.13

C:\Users\akash.jindal>npm install -g json-server
C:\Users\akash.jindal\AppData\Roaming\npm\json-server -> C:\Users\akash.jindal\AppData\Roaming\npm\node_modules\json-server\lib\cli\bin.js
+ json-server@0.16.3
added 182 packages from 80 contributors in 62.098s

C:\Users\akash.jindal>json-server --version
0.16.3

C:\Users\akash.jindal>_
```

12. We will create a json file for having our json data :

Json data → key : value relationship

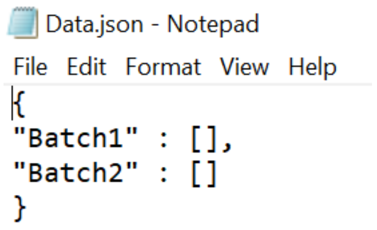
where :

[] denotes **Array Notation**

{} denotes **Object Notation**

13. Create a **.json file** using txt file with the data as shown :

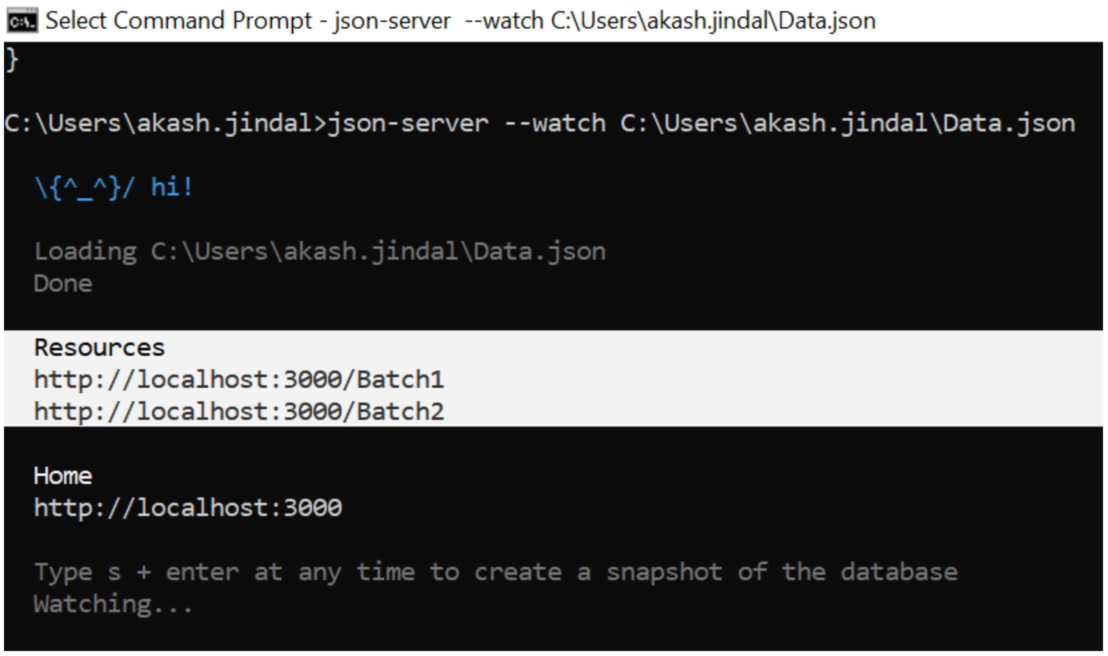
Data.json



```
{
"Batch1" : [],
"Batch2" : []
}
```

14. Start json-server under cmd :

command : json-server –watch 'path of your json file'


Select Command Prompt - json-server --watch C:\Users\akash.jindal\Data.json

```
}

C:\Users\akash.jindal>json-server --watch C:\Users\akash.jindal\Data.json

  \{^_^}/ hi!

  Loading C:\Users\akash.jindal\Data.json
  Done

  Resources
  http://localhost:3000/Batch1
  http://localhost:3000/Batch2

  Home
  http://localhost:3000

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

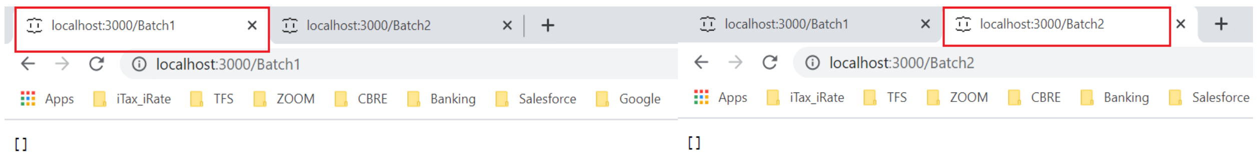
Resources :

http://localhost:3000/Batch1
http://localhost:3000/Batch2

will be our dummy API's or the individual features or functionality or resources.

(use CTRL + C to stop the json-server)

15. Verify the same using browser's tab :



```
[]
```

```
[]
```

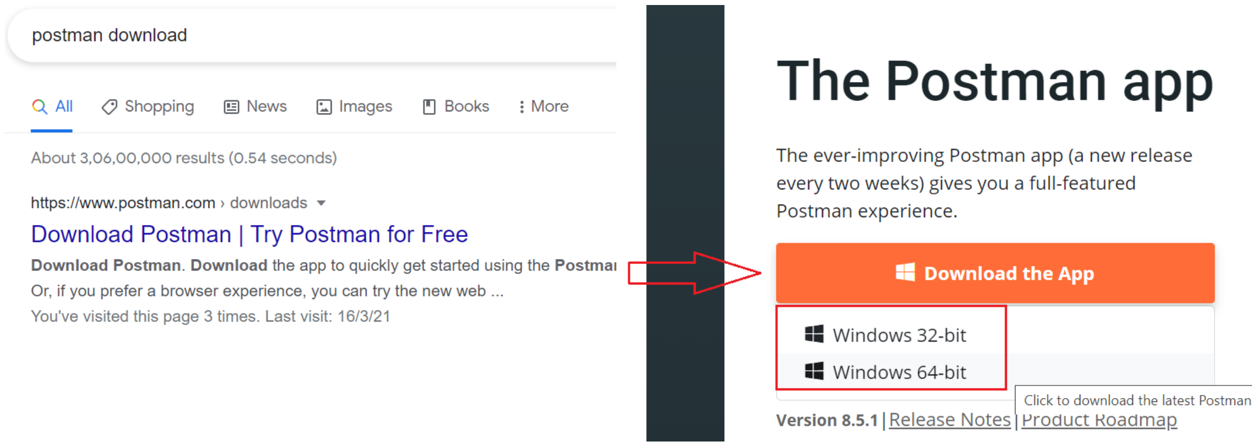
We are going to insert the data under these API's.

POSTMAN

POSTMAN is a tool used for REST API testing manually or through automation.

Getting Started with POSTMAN :

1. Open a browser and download POSTMAN :


postman download

About 3,06,00,000 results (0.54 seconds)

https://www.postman.com › downloads

Download Postman | Try Postman for Free

Download Postman. **Download** the app to quickly get started using the **Postman**
Or, if you prefer a browser experience, you can try the new web ...
You've visited this page 3 times. Last visit: 16/3/21

The Postman app

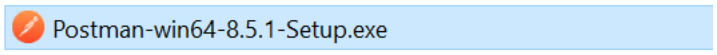
The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

Click to download the latest Postman

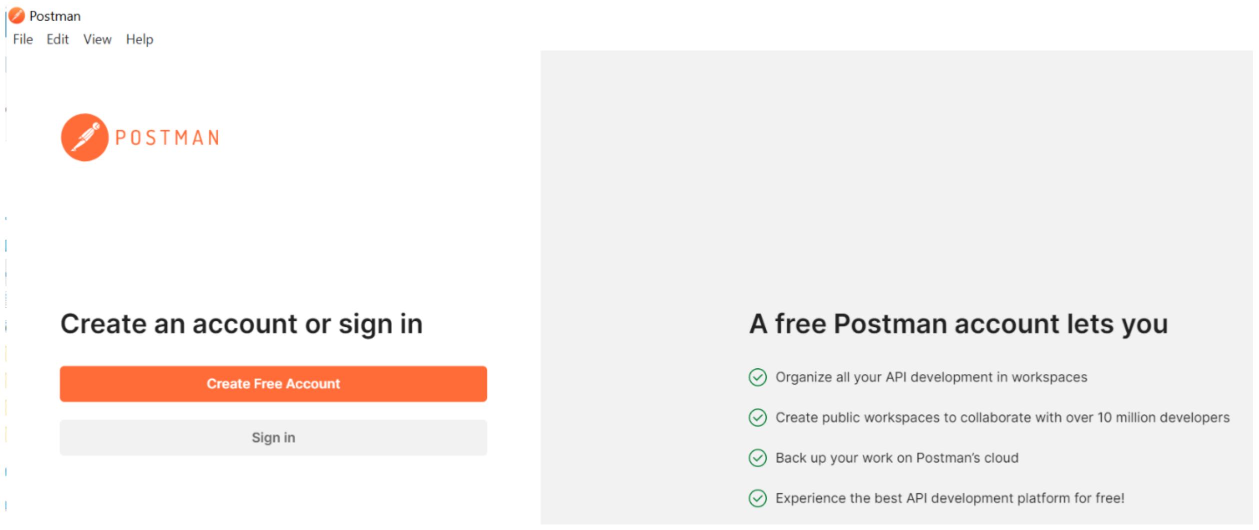
Version 8.5.1 | Release Notes | Product Roadmap

Download the application meeting your system requirement.

2. Downloaded file will be an executable file :


Postman-win64-8.5.1-Setup.exe

3. Simply click on the setup and install.



Postman
File Edit View Help

POSTMAN

Create an account or sign in

Create Free Account

Sign in

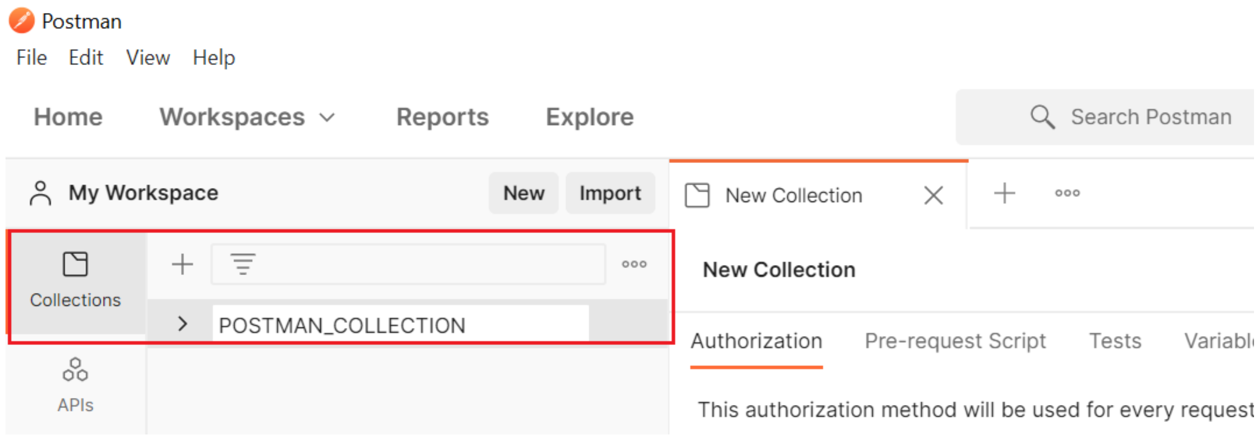
A free Postman account lets you

- Organize all your API development in workspaces
- Create public workspaces to collaborate with over 10 million developers
- Back up your work on Postman's cloud
- Experience the best API development platform for free!

4. Create a POSTMAN account and sign in with same.

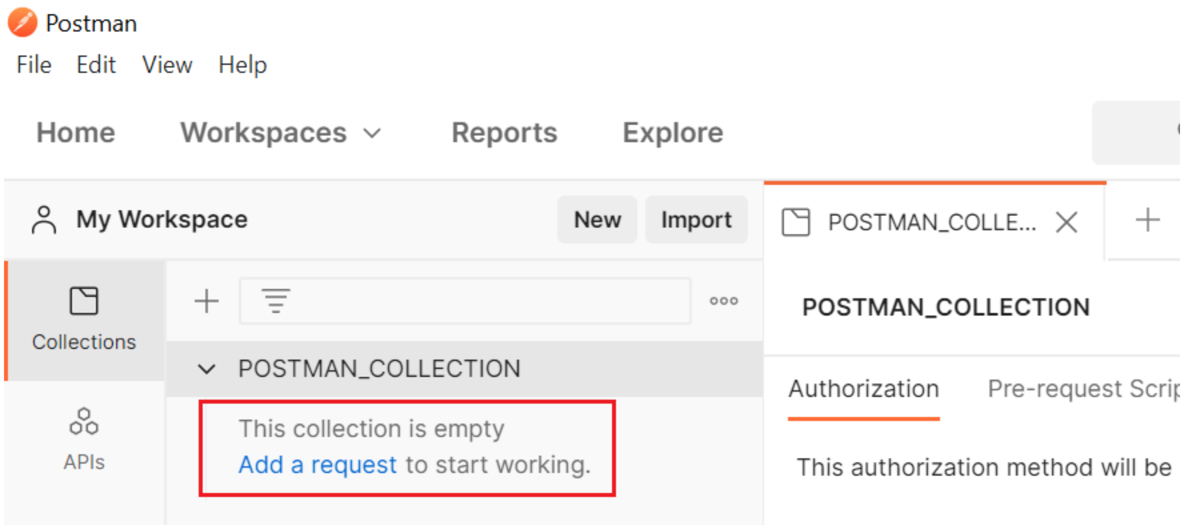
5. Create a **COLLECTIONS** in POSTMAN :

POSTMAN COLLECTIONS are a group of saved requests you can organize into folders.



This authorization method will be used for every request

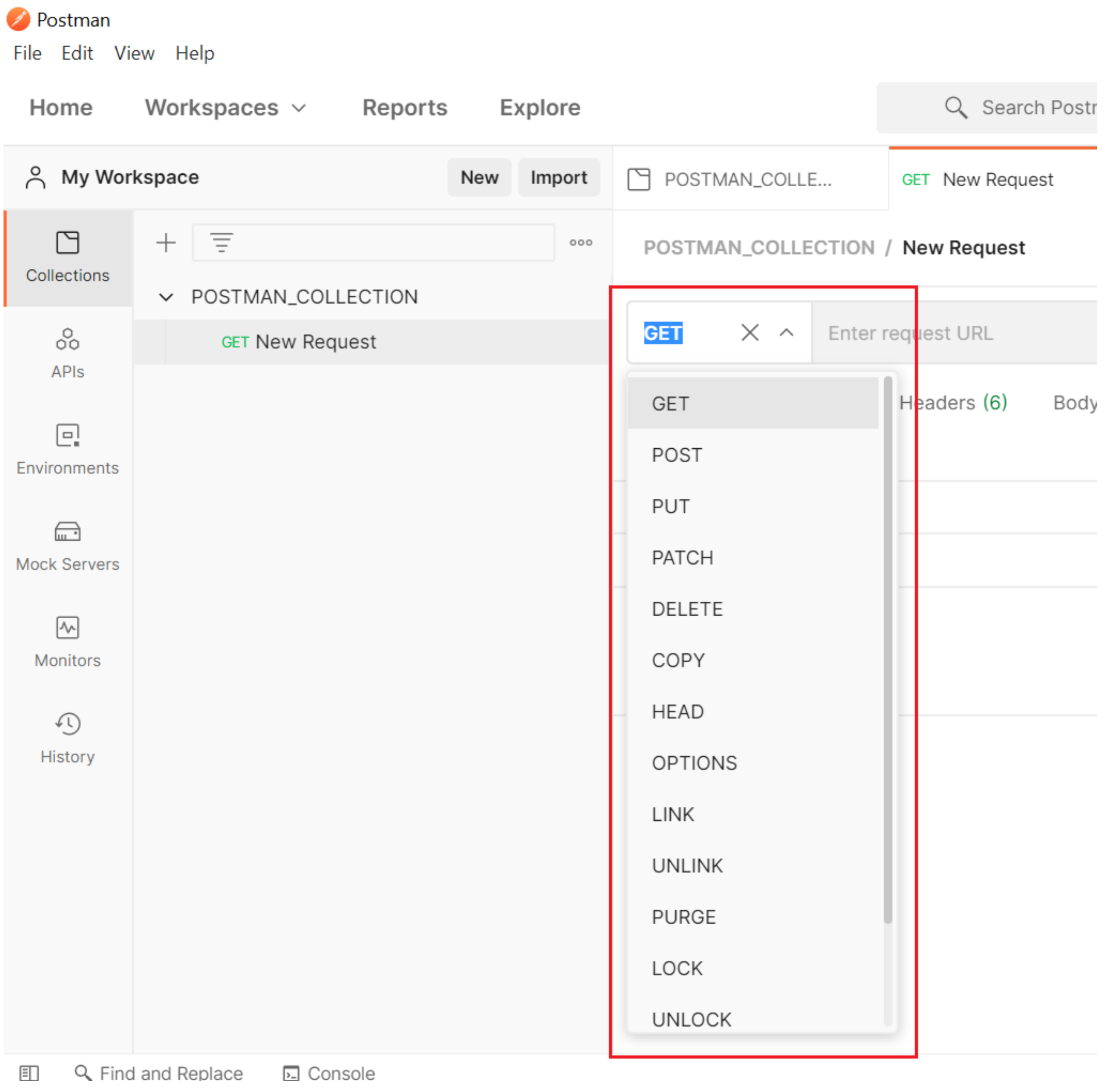
6. Click **Add a Request** to start using POSTMAN for different **HTTP Requests** :



This collection is empty
Add a request to start working.

This authorization method will be

7. POSTMAN will display the list of different HTTP Requests :



Hitting HTTP REQUESTS Using POSTMAN

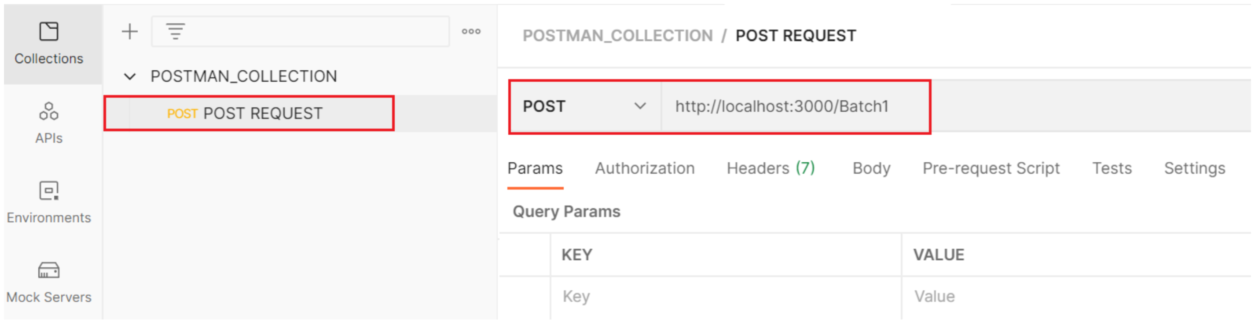
1. POST REQUEST using POSTMAN :

Create a new request under :

COLLECTIONS → POST REQUEST

Select **HTTP Request as POST** and enter the dunny API (one we created using json-server) :

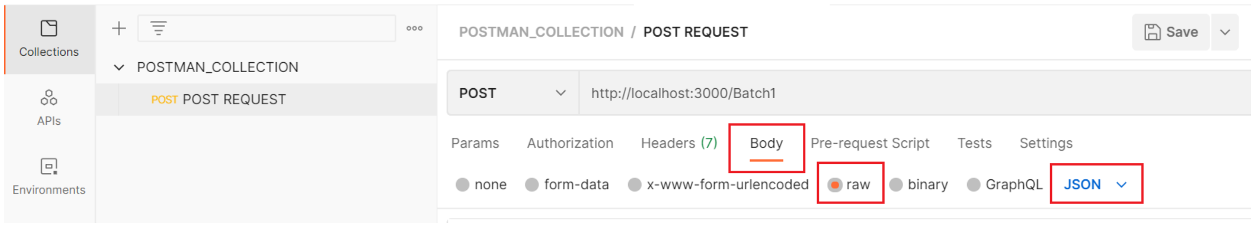
http://localhost:3000/Batch1



POSTMAN_COLLECTION / POST REQUEST

POST http://localhost:3000/Batch1

Select **Body → raw** and **JSON as parameters for json body data** :



POSTMAN_COLLECTION / POST REQUEST

POST http://localhost:3000/Batch1

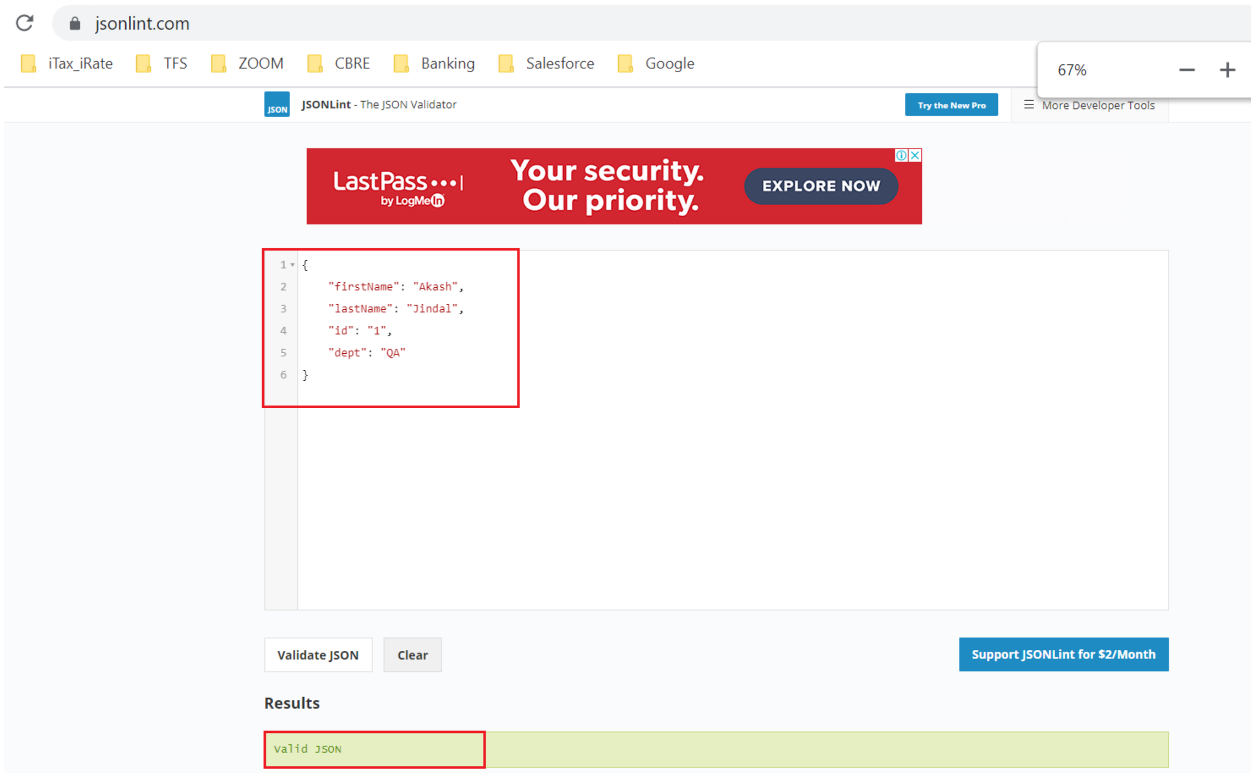
Body: ● raw, JSON

A. POST REQUEST : SIMPLE JSON :

Simple JSON DATA :

```
{
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "1",
        "dept": "QA"
}
```

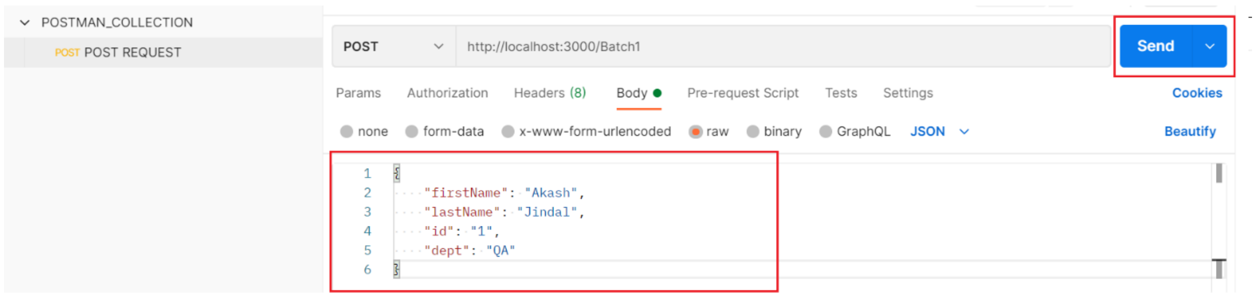
We can make use of **https://jsonlint.com** to validate the JSON data :



```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
}
```

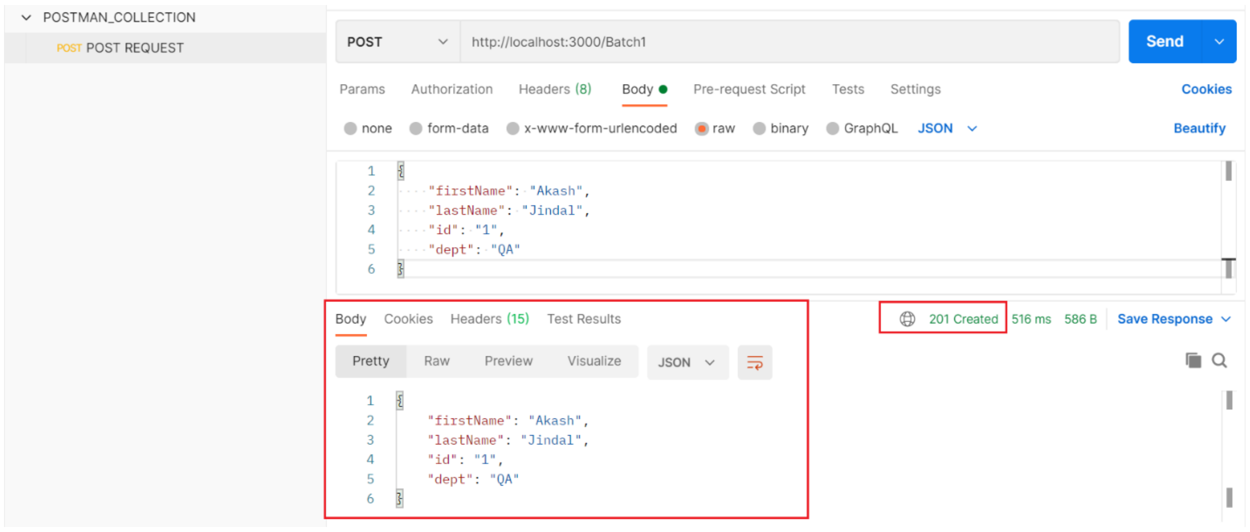
```
valid JSON
```

Paste the data under the **Body** in POSTMAN and click **Send** :



```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
}
```

Validate the **Status Code** and the received data :



POSTMAN_COLLECTION
POST POST REQUEST
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify
Body Cookies Headers (15) Test Results
Pretty Raw Preview Visualize JSON
Save Response

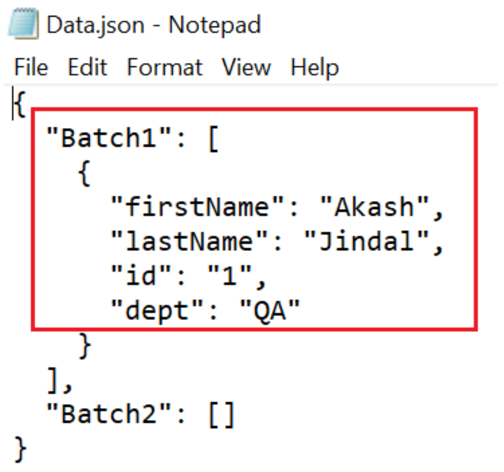
POST http://localhost:3000/Batch1

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
}
```

201 Created 516 ms 586 B

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
}
```

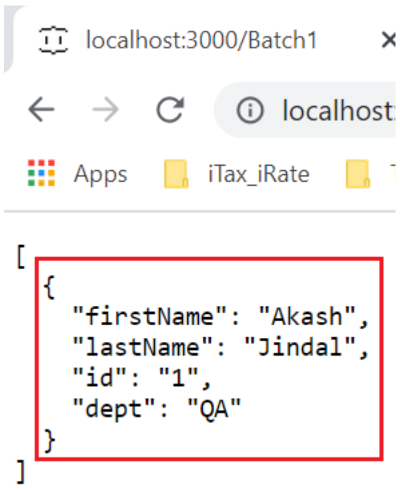
Validate the same under the **Data.json** file :



```
{
  "Batch1": [
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "1",
      "dept": "QA"
    }
  ],
  "Batch2": []
}
```

Validate the same using browser's tab :

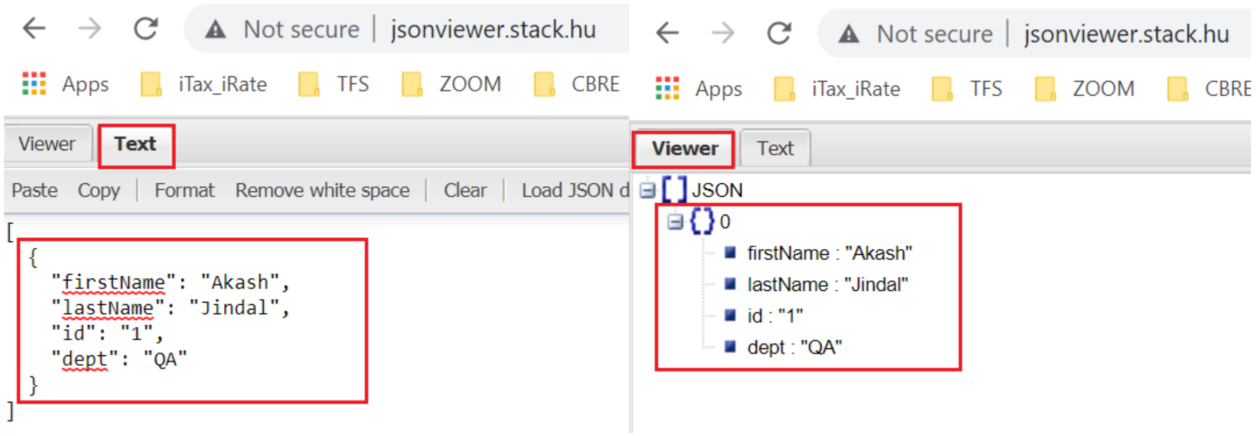
Make sure to use the same API used for POST Request :



```
[
{
  "firstName": "Akash",
  "lastName": "Jindal",
  "id": "1",
  "dept": "QA"
}
]
```

Validate the json structure using **http://jsonviewer.stack.hu** :

Paste the json data and validate :



```
[
 {
  "firstName": "Akash",
  "lastName": "Jindal",
  "id": "1",
  "dept": "QA"
 }
]
```

```
firstName : "Akash"
lastName : "Jindal"
id : "1"
dept : "QA"
```

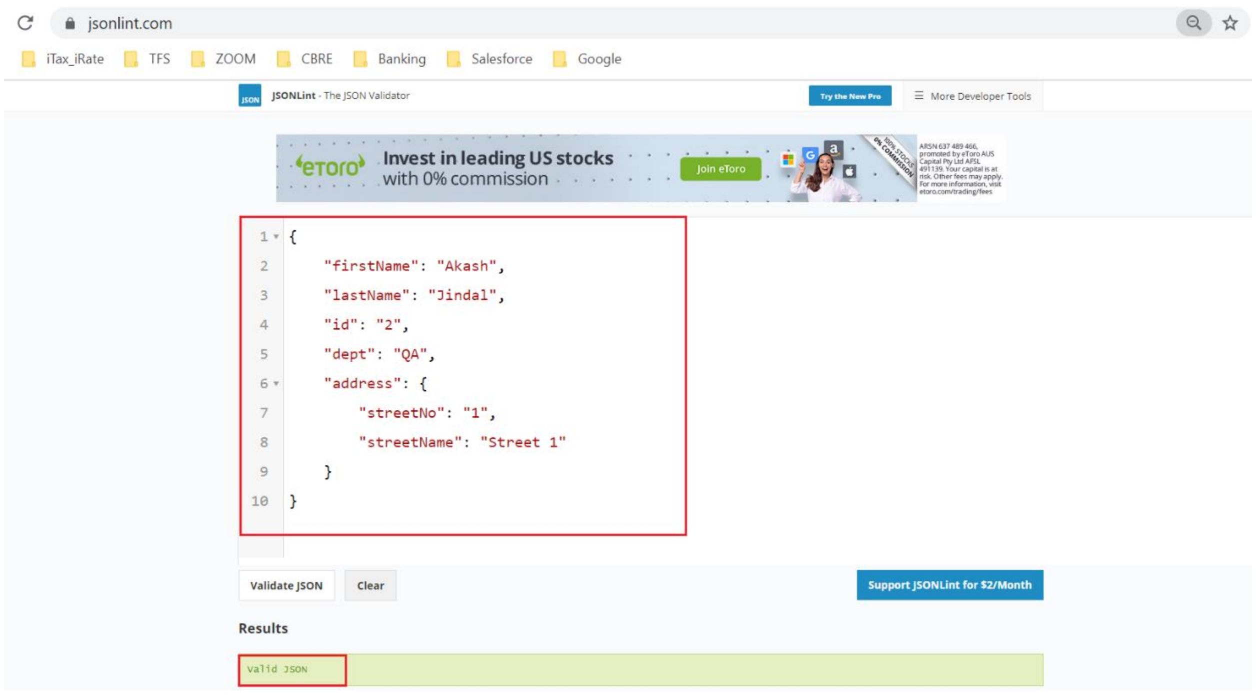
(Please do note that we must make use of "id" which is case-sensitive and unique while using the data to have unique entry of the records just like PRIMARY KEY else a random id will be generated while hitting HTTP requests)

B. POST REQUEST : COMPLEX JSON :

Complex JSON DATA :

```
{
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "2",
        "dept": "QA",
        "address" :  {
                "streetNo": "1",
                "streetName": "Street 1"
        }
}
```

We can make use of **https://jsonlint.com** to validate the JSON data :

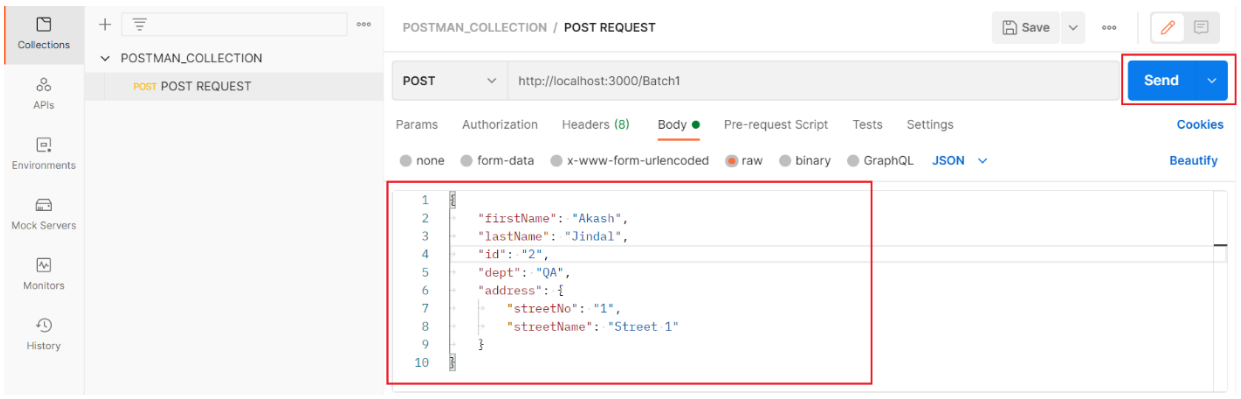


```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
}
```

Results

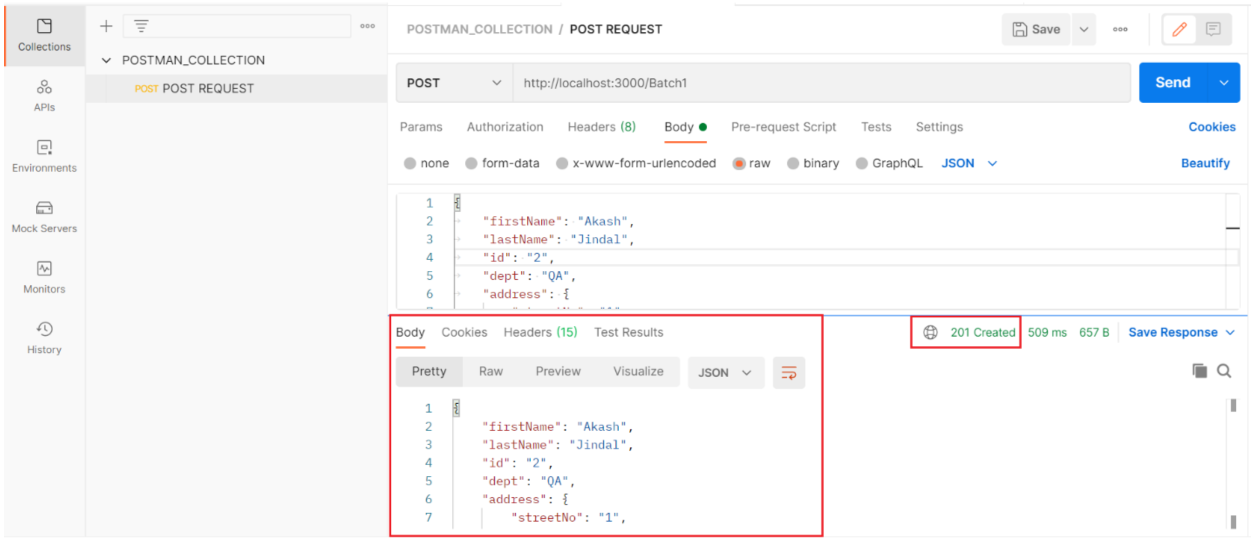
```
valid JSON
```

Paste the data under the **Body** in POSTMAN and click **Send** :



```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
}
```

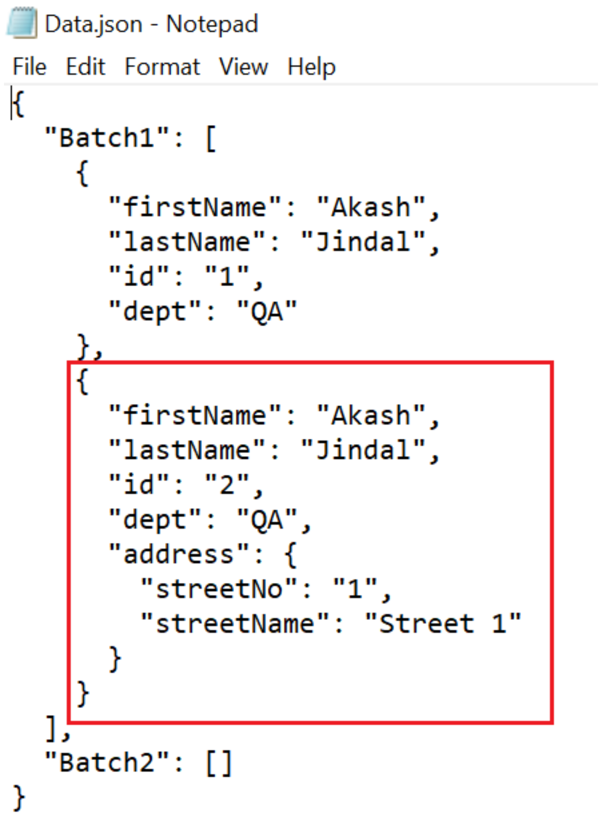
Validate the **Status Code** and the received data :



```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
```

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
        "streetNo": "1",
```

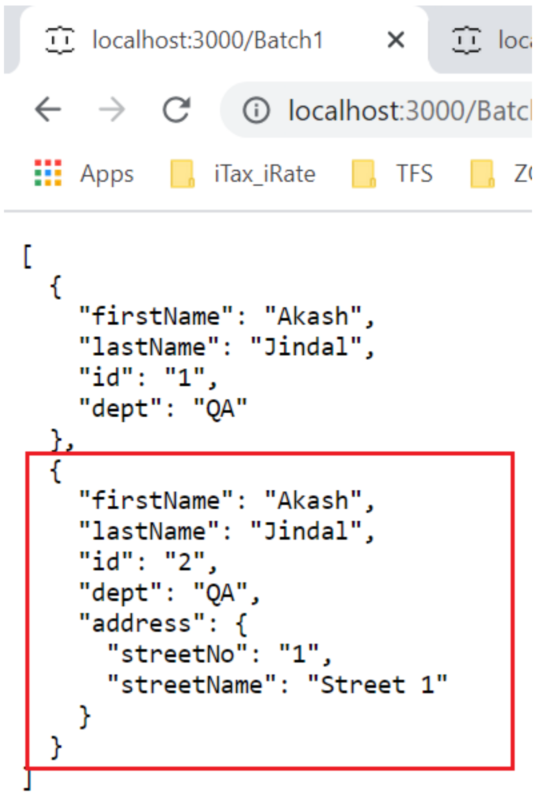
Validate the same under the **Data.json** file :



```
{
  "Batch1": [
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "1",
      "dept": "QA"
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "2",
      "dept": "QA",
      "address": {
        "streetNo": "1",
        "streetName": "Street 1"
      }
    }
  ],
  "Batch2": []
}
```

Validate the same using browser’s tab :

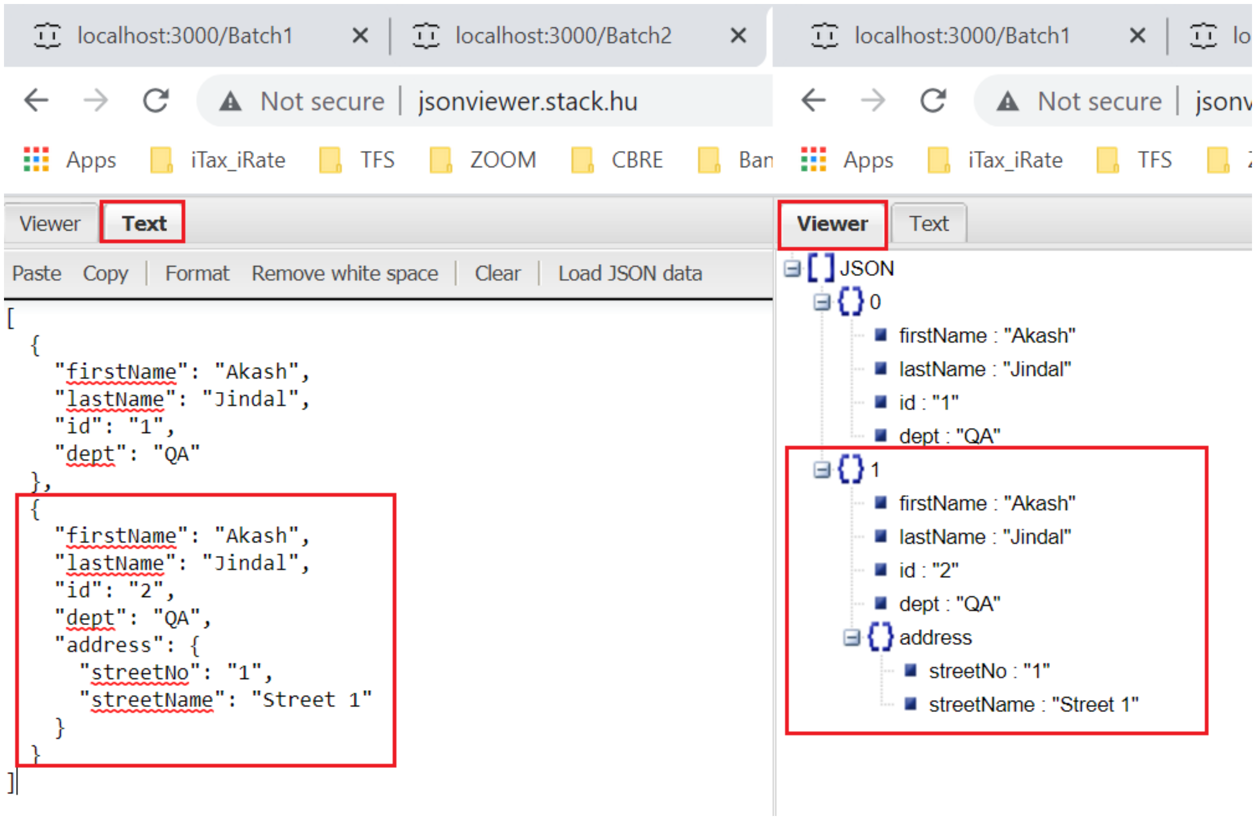
Make sure to use the same API used for POST Request :



```
[
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
  },
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    }
  }
]
```

Validate the json structure using **http://jsonviewer.stack.hu** :

Paste the json data and validate :



```
[
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
  },
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    }
  }
]
```

```
[ ] JSON
  { } 0
    firstName : "Akash"
    lastName : "Jindal"
    id : "1"
    dept : "QA"
  { } 1
    firstName : "Akash"
    lastName : "Jindal"
    id : "2"
    dept : "QA"
    { } address
      streetNo : "1"
      streetName : "Street 1"
```

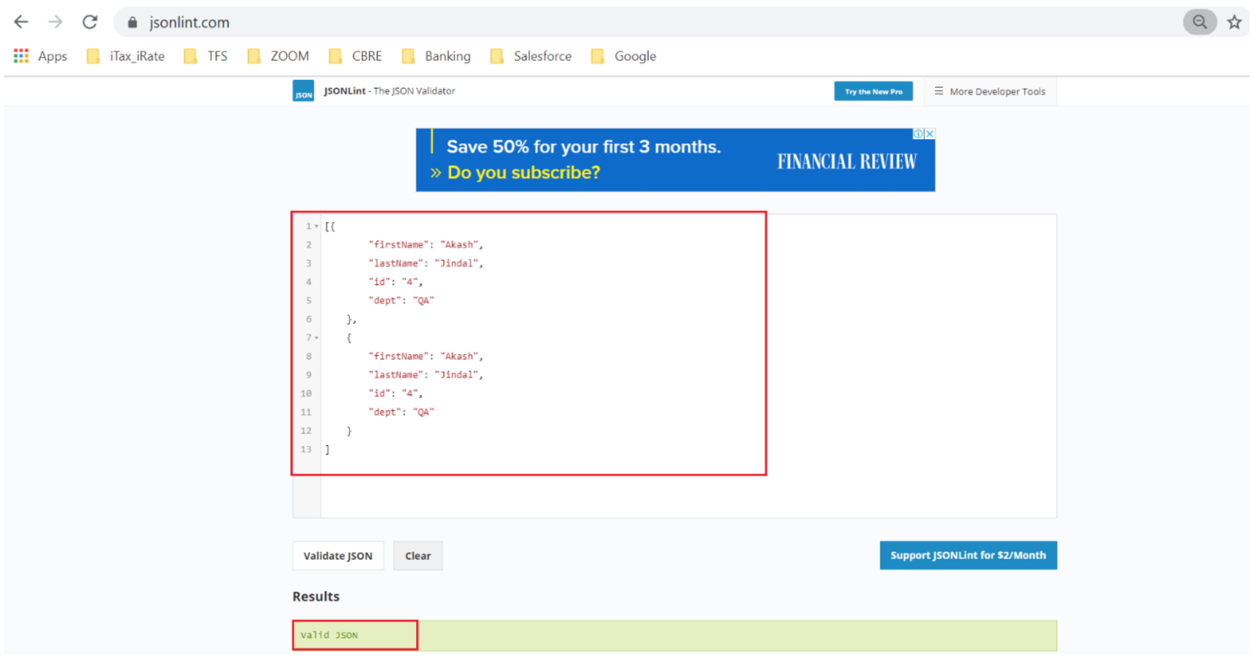
(Please do note that we must make use of "id" which is case-sensitive and unique while using the data to have unique entry of the records just like PRIMARY KEY else a random id will be generated while hitting HTTP requests)

C. POST REQUEST : IN FORM OF ARRAY JSON :

In Form Of Array JSON DATA :

```
[
{
                "firstName": "Akash",
                "lastName": "Jindal",
                "id": "4",
                "dept": "QA"
        },
        {
                "firstName": "Akash",
                "lastName": "Jindal",
                "id": "4",
                "dept": "QA"
        }
]
```

We can make use of **https://jsonlint.com** to validate the JSON data :



jsonlint.com

Apps iTax_iRate TFS ZOOM CBRE Banking Salesforce Google

JSONLint - The JSON Validator

Try the New Pro

More Developer Tools

Save 50% for your first 3 months.
» Do you subscribe?
FINANCIAL REVIEW

```
[{
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
    },
    {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
    }
]
```

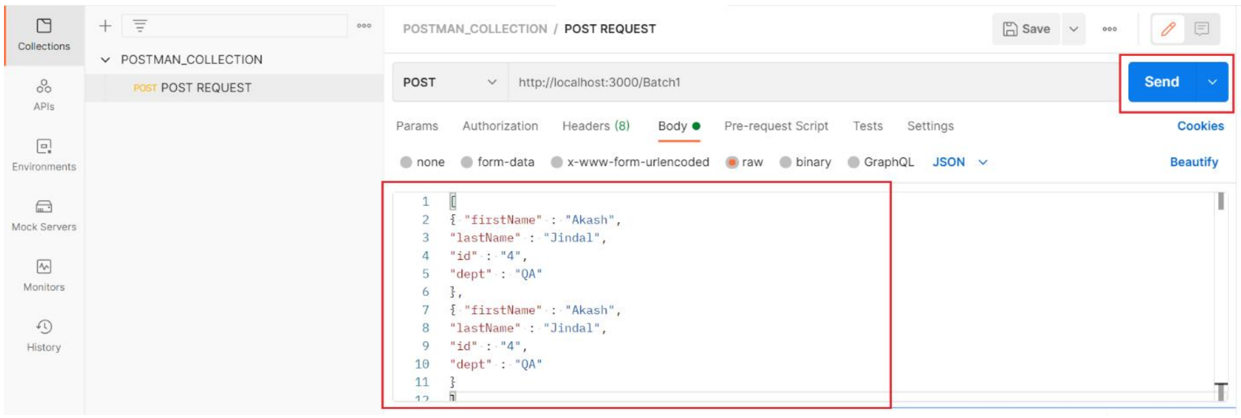
Validate JSON Clear

Support JSONLint for $2/Month

Results

```
Valid JSON
```

Paste the data under the Body in **POSTMAN** and click **Send** :

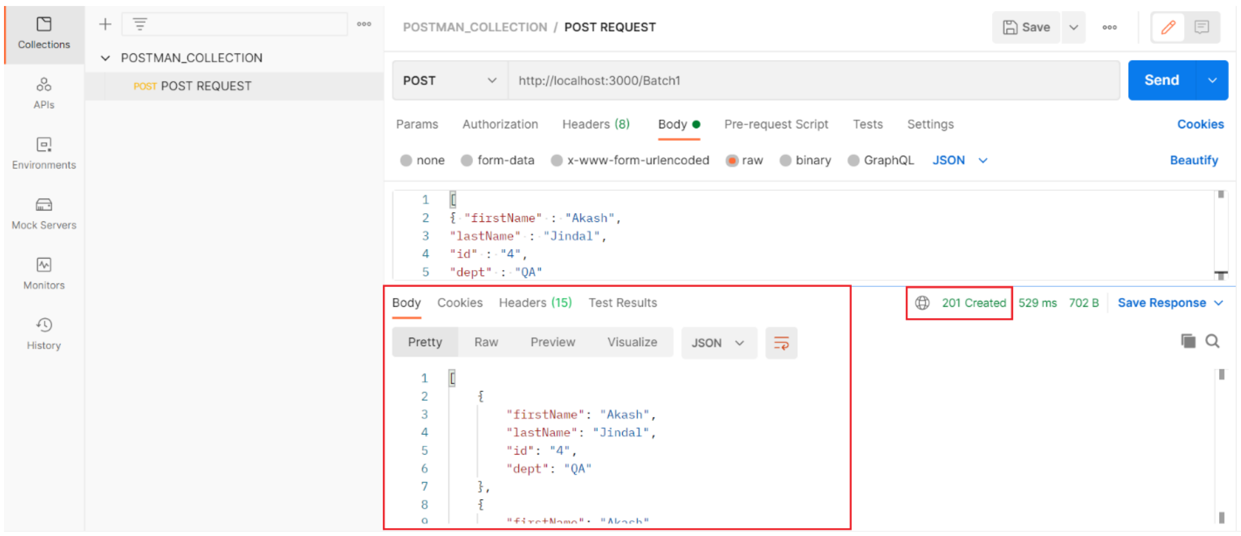

POSTMAN_COLLECTION / POST REQUEST

POST http://localhost:3000/Batch1

Body: raw JSON

```
[
{ "firstName" : "Akash",
"lastName" : "Jindal",
"id" : "4",
"dept" : "QA"
},
{ "firstName" : "Akash",
"lastName" : "Jindal",
"id" : "4",
"dept" : "QA"
}
]
```

Validate the **Status Code** and the received data :

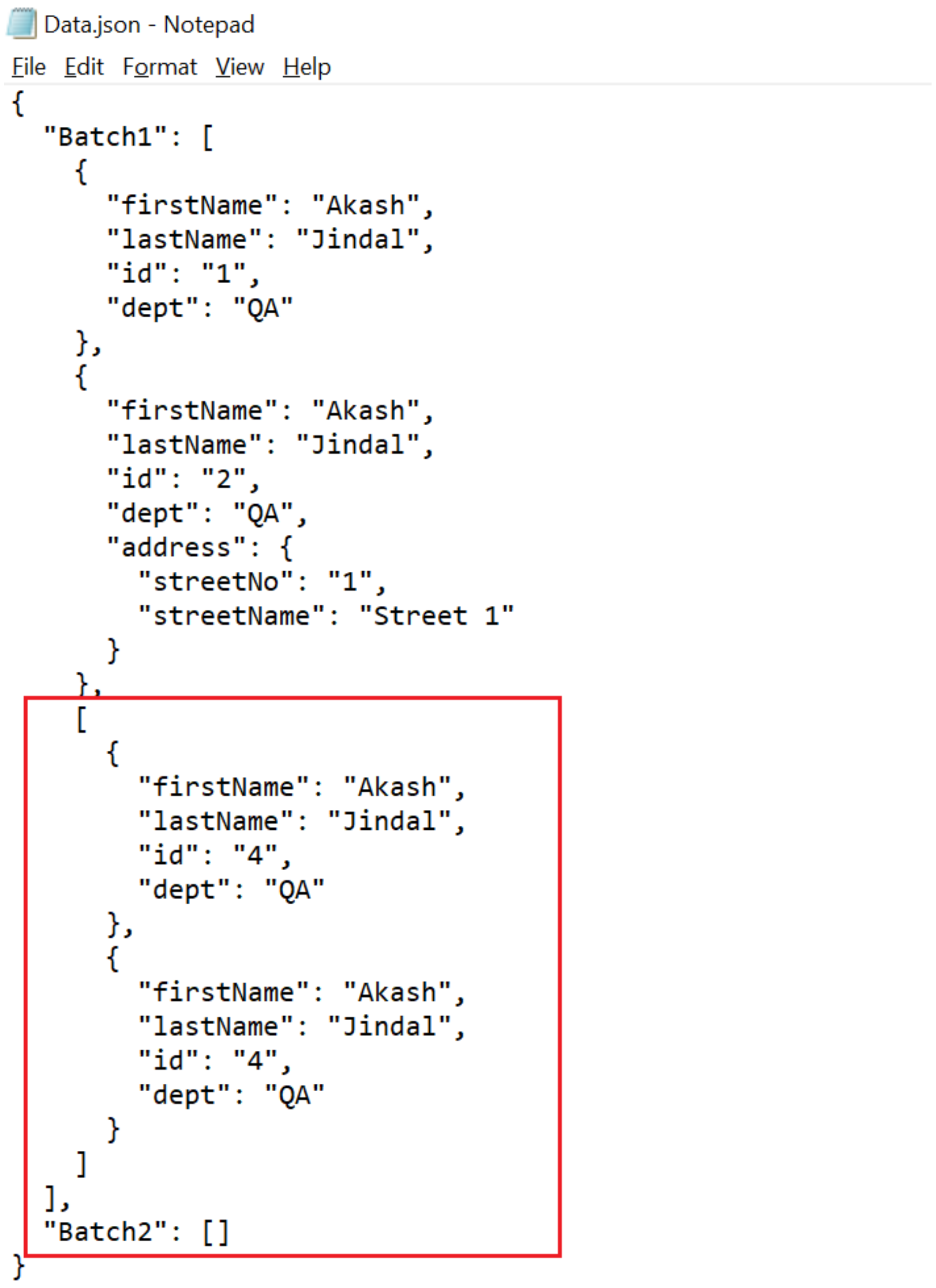

POSTMAN_COLLECTION / POST REQUEST

POST http://localhost:3000/Batch1

201 Created 529 ms 702 B

```
[
    {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
    },
    {
        "firstName": "Akash"
```

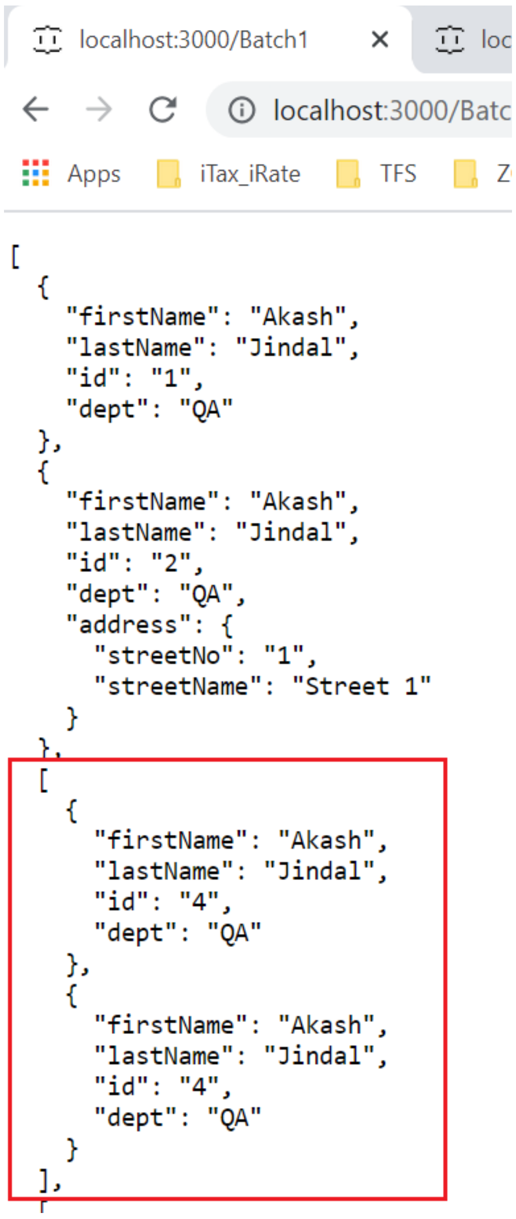
Validate the same under the **Data.json** file :



```
{
  "Batch1": [
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "1",
      "dept": "QA"
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "2",
      "dept": "QA",
      "address": {
        "streetNo": "1",
        "streetName": "Street 1"
      }
    },
    [
      {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      },
      {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      }
    ]
  ],
  "Batch2": []
}
```

Validate the same using browser's tab :

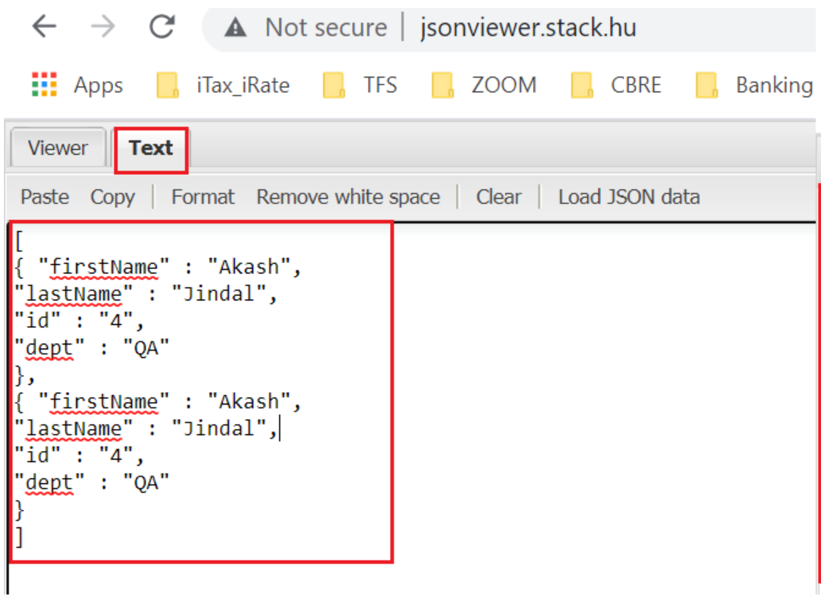
Make sure to use the same API used for POST Request :



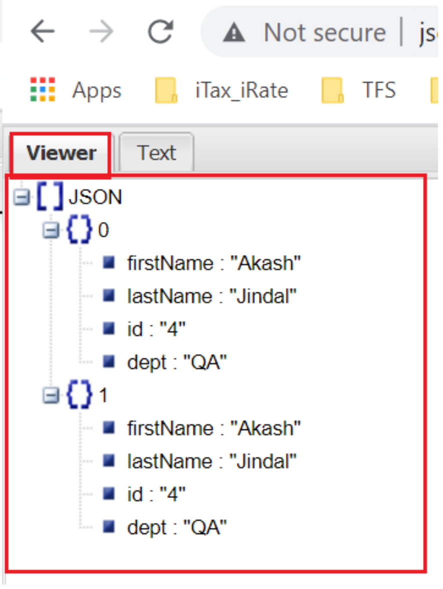
```
[
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
  },
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    }
  },
  [
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    }
  ],
```

Validate the json structure using **http://jsonviewer.stack.hu** :

Paste the json data and validate :


Not secure | jsonviewer.stack.hu
Apps iTax_iRate TFS ZOOM CBRE Banking
Viewer Text
Paste Copy Format Remove white space Clear Load JSON data

```
[
{ "firstName" : "Akash",
"lastName" : "Jindal",
"id" : "4",
"dept" : "QA"
},
{ "firstName" : "Akash",
"lastName" : "Jindal",
"id" : "4",
"dept" : "QA"
}
]
```



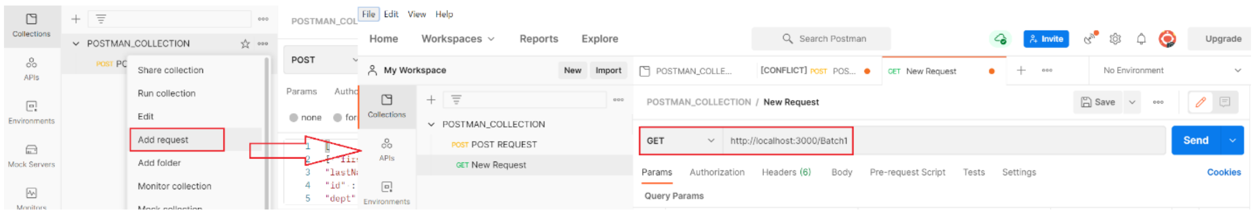
```
[ ] JSON
  { } 0
    firstName : "Akash"
    lastName : "Jindal"
    id : "4"
    dept : "QA"
  { } 1
    firstName : "Akash"
    lastName : "Jindal"
    id : "4"
    dept : "QA"
```

(Please do note that we must make use of "id" which is case-sensitive and unique while using the data to have unique entry of the records just like PRIMARY KEY else a random id will be generated while hitting HTTP requests)

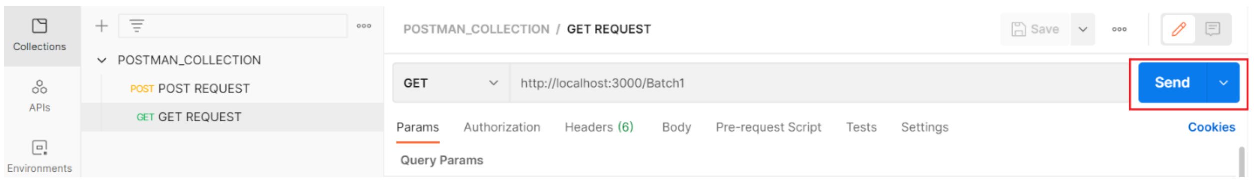
2. **GET REQUEST** using POSTMAN :

Create a new request under COLLECTIONS.

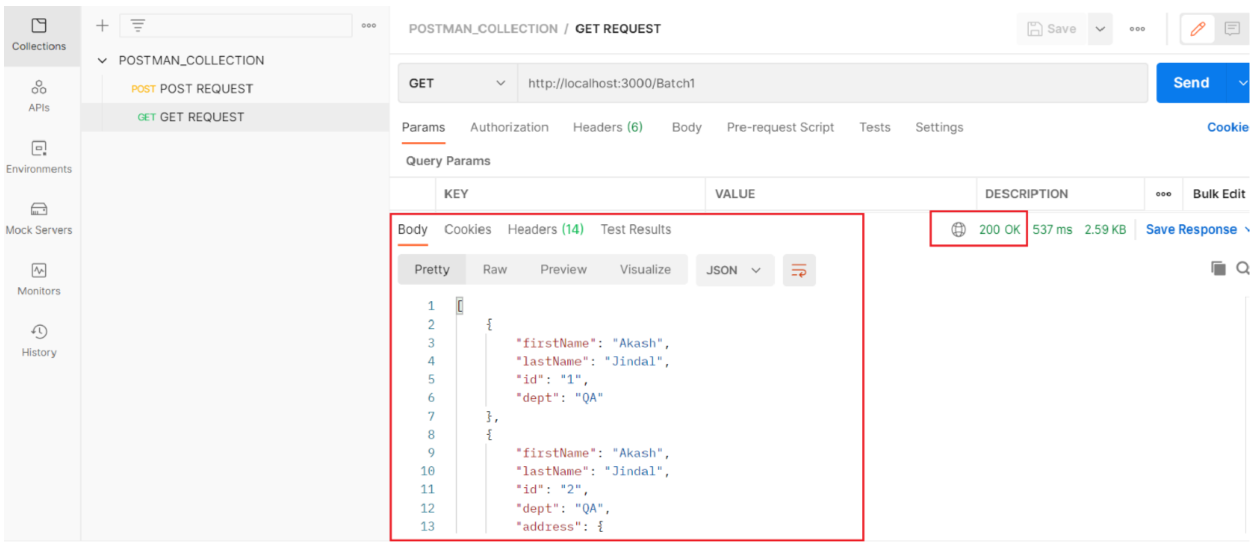
Select **GET request from the HTTP Requests list** and paste the required API.



Click **Send**.



Validate the **Status Code** and the received data :



```
[
    {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "1",
        "dept": "QA"
    },
    {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "2",
        "dept": "QA",
        "address": {
```

Result set will be the complete list of data under the specified API as we have not used any **Query Parameter** or **Path Parameter** along with the API.

Query Parameter : Query Parameters are used to control what data is returned in endpoint responses, **denoted by "?"**.

Example :

Use **URI** as : http://localhost:3000/Batch1?firstaName=Akash

Here,

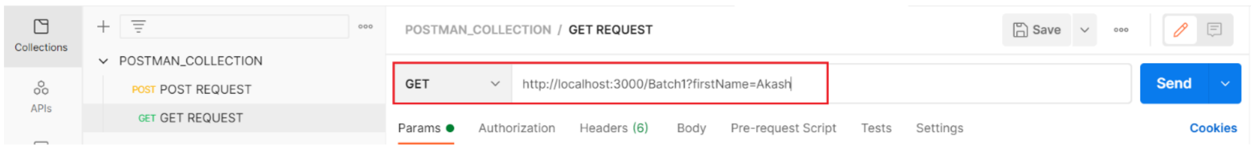
URI : Base + Endpoint + Parameter

Base : http://localhost:3000

Endpoint : /Batch1

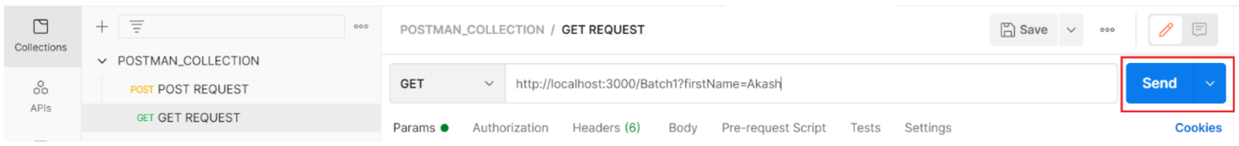
Query Parameter : ?firstName=Akash

Select GET request from the HTTP Requests list and paste the required API.



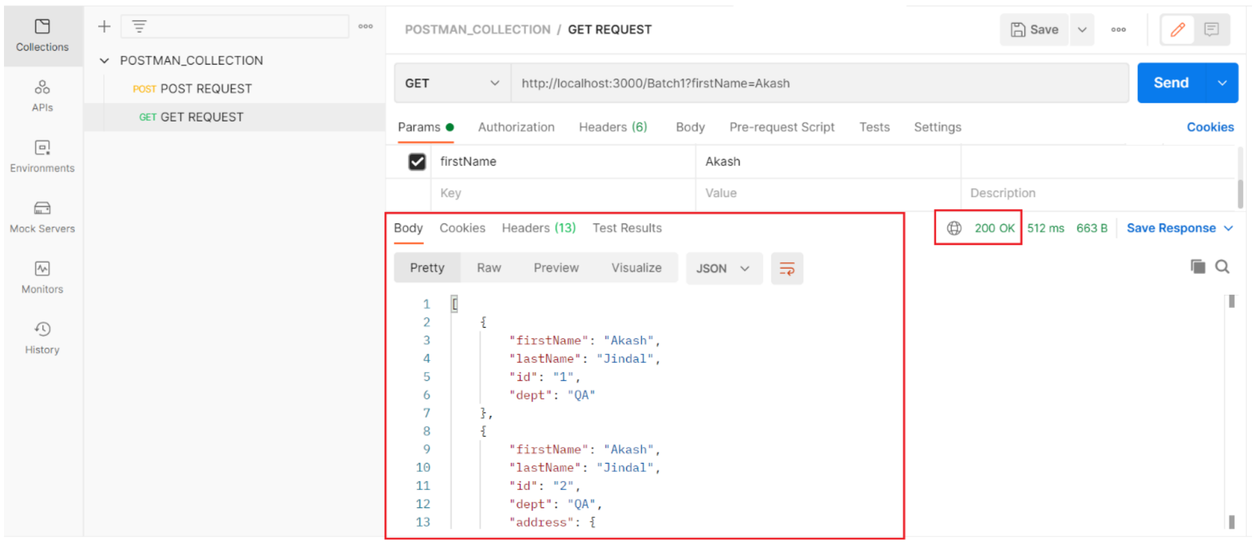
GET http://localhost:3000/Batch1?firstName=Akash

Click **Send**.



GET http://localhost:3000/Batch1?firstName=Akash

Validate the **Status Code** and the received data :



GET http://localhost:3000/Batch1?firstName=Akash

	Key	Value	Description
✓	firstName	Akash	

200 OK 512 ms 663 B

```
[
    {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "1",
        "dept": "QA"
    },
    {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "2",
        "dept": "QA",
        "address": {
```

Result set will retrieve the data associated with **firstName : Akash**.

Path Parameter : Path Parameters are used to identify a resource uniquely, **denoted by "/"**.

Example :

Use **URI** as : http://localhost:3000/Batch1\1

Here,

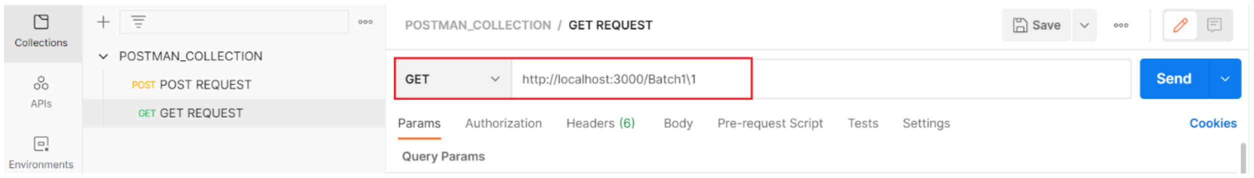
URI : Base + Endpoint + Parameter

Base : http://localhost:3000

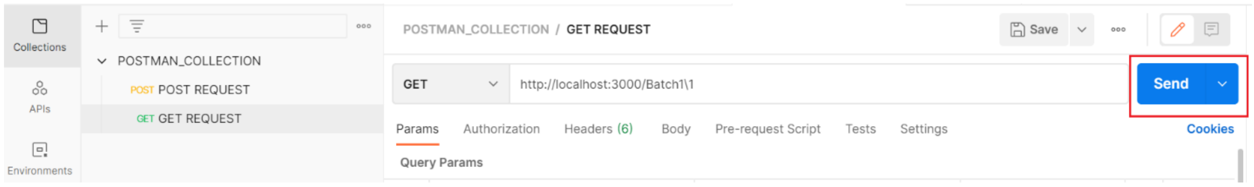
Endpoint : /Batch1

Path Parameter : /1

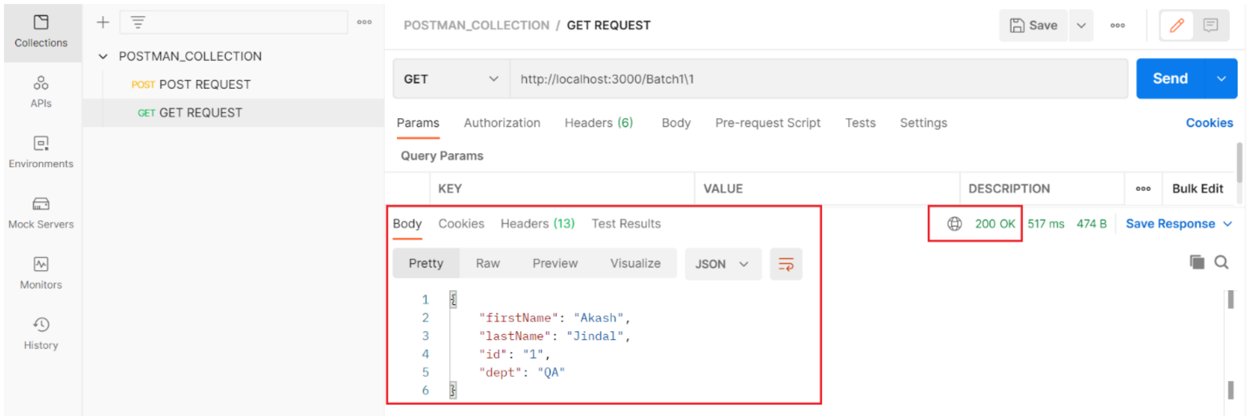
Select **GET request from the HTTP Requests list** and paste the required API.



Click **Send**.



Validate the **Status Code** and the received data :



```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA"
}
```

Result set will retrieve the data associated with **id : 1**.

3. DELETE REQUEST using POSTMAN :

Use **URI** as : http://localhost:3000/Batch1\2

Here,

URI : Base + Endpoint + Parameter

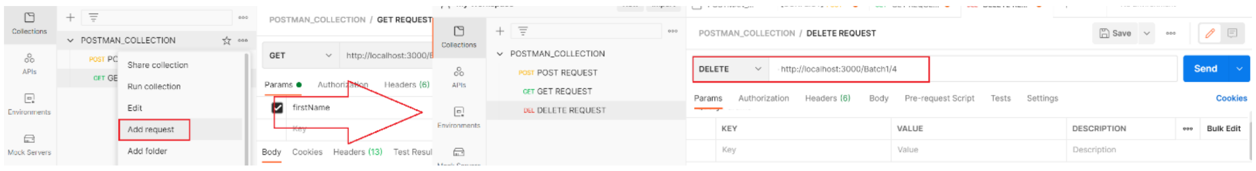
Base : http://localhost:3000

Endpoint : /Batch1

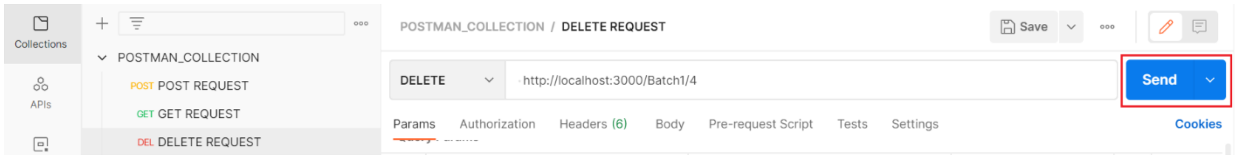
Path Parameter : /2

Create a new request under COLLECTIONS.

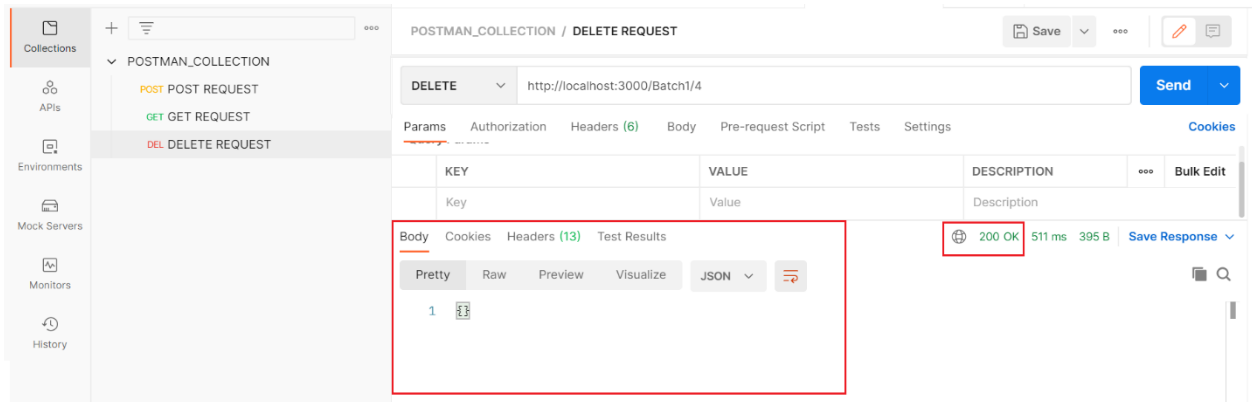
Select **GET request from the HTTP Requests list** and paste the required API.



Click **Send**.



Validate the **Status Code** and the received data :



```
1 {}
```

Executing the command will delete all the records having **id : 4**.

4. PUT REQUEST using POSTMAN :

Use **URI** as : http://localhost:3000/Batch1\2

Here,

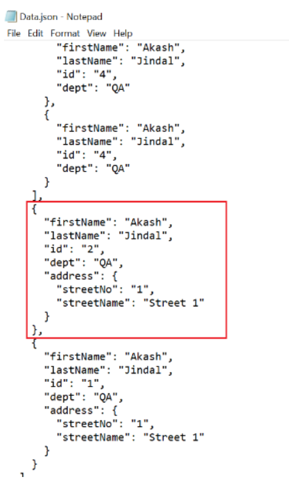
URI : Base + Endpoint + Parameter

Base : http://localhost:3000

Endpoint : /Batch1

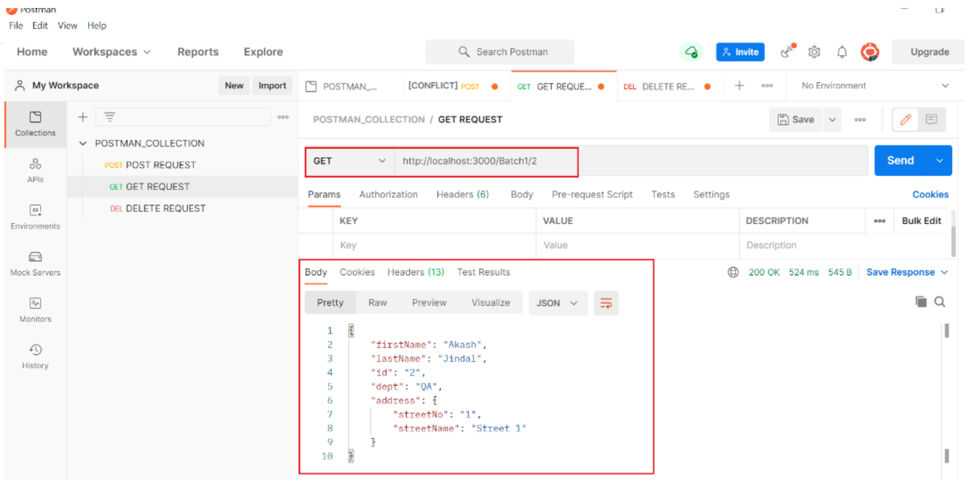
Path Parameter : /2

First, have a look at the existing data associated with **id : 2** under **Data.json** file and **POSTMAN** :



Data.json - Notepad

```
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    }
  ],
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    }
  },
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "1",
    "dept": "QA",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    }
  }
```



POSTMAN_COLLECTION / GET REQUEST

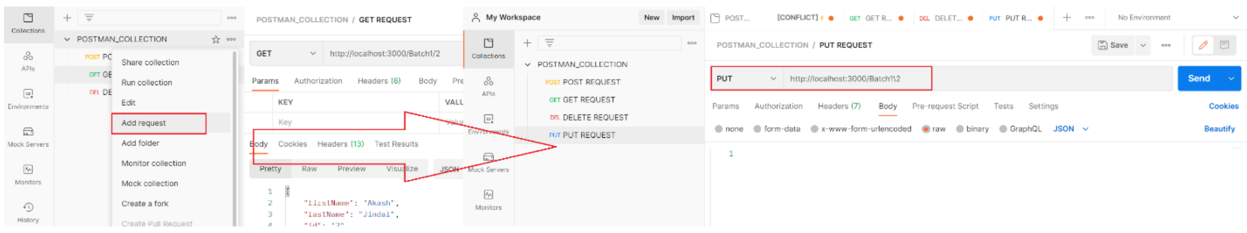
GET http://localhost:3000/Batch1/2

200 OK 524 ms 545 B

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "dept": "QA",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
}
```

Create a new request under COLLECTIONS.

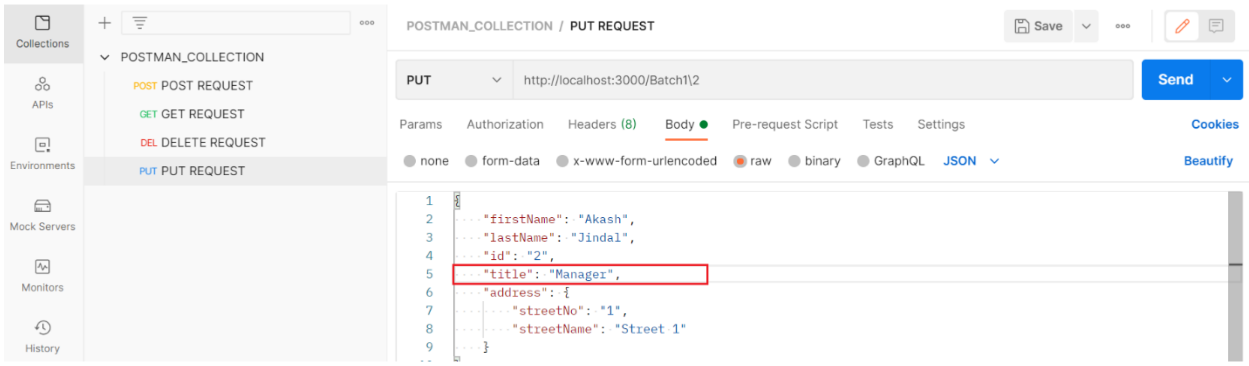
Select **PUT request from the HTTP Requests list** and paste the required API.



POSTMAN_COLLECTION / PUT REQUEST

PUT http://localhost:3000/Batch1\2

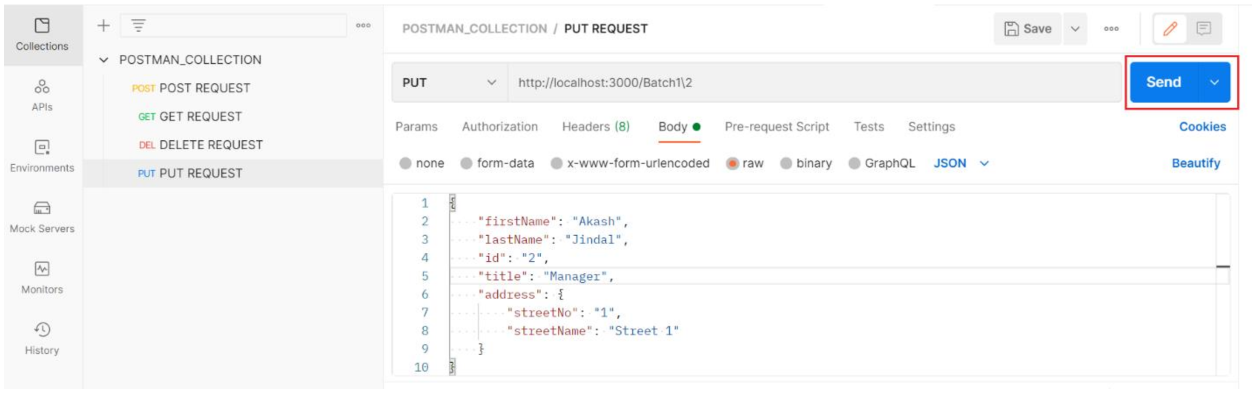
Update the required data : **from : dept : QA to : title : Manager** :



PUT http://localhost:3000/Batch1\2

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
```

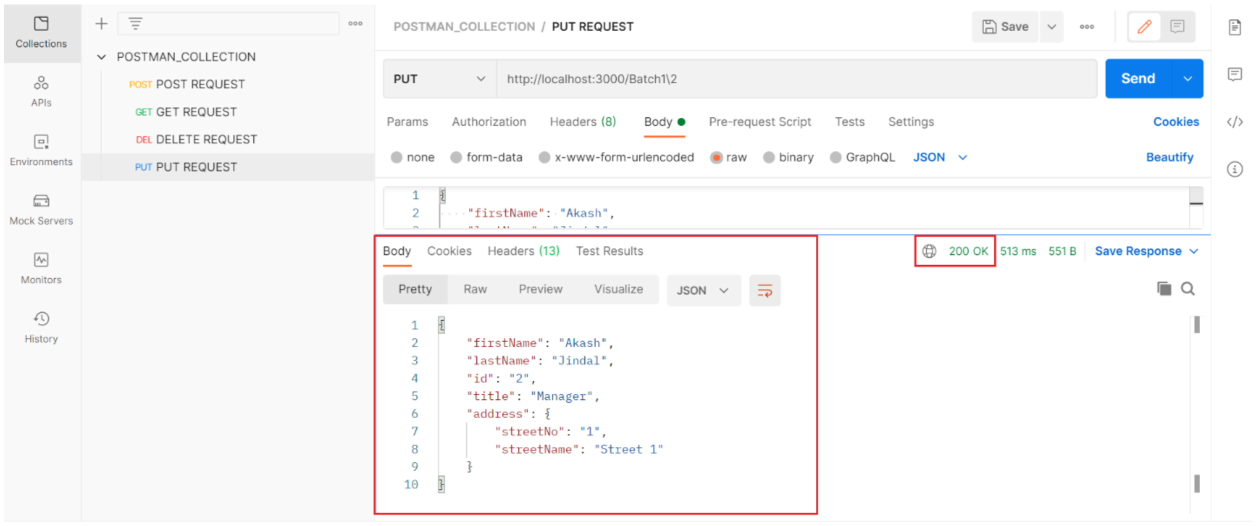
Click **Send**.



PUT http://localhost:3000/Batch1\2

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
}
```

Validate the **Status Code** and the received data :



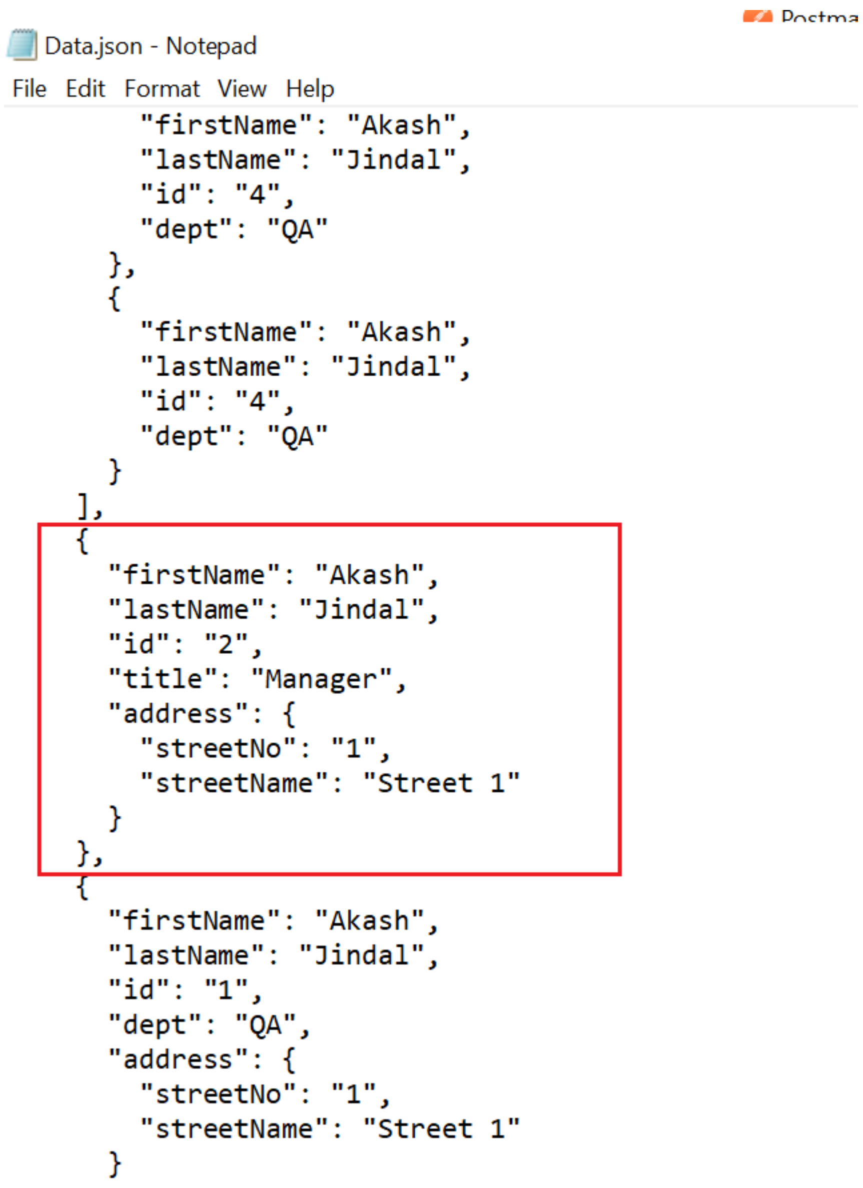
PUT http://localhost:3000/Batch1\2

200 OK 513 ms 551 B

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
}
```

Executing the command will update the record having **id : 2**.

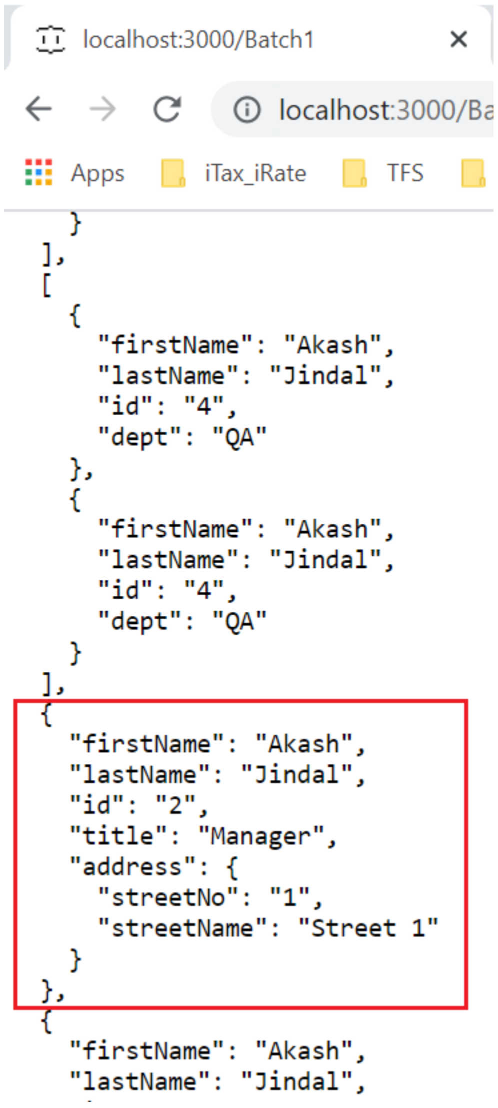
Validate the same under the **Data.json** file :



```
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      },
      {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      }
    ],
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "2",
      "title": "Manager",
      "address": {
        "streetNo": "1",
        "streetName": "Street 1"
      }
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "1",
      "dept": "QA",
      "address": {
        "streetNo": "1",
        "streetName": "Street 1"
      }
```

Validate the same using browser's tab :

Make sure to use the same API used for PUT Request :



```
    }
  ],
  [
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    }
  ],
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    }
  },
  {
    "firstName": "Akash",
    "lastName": "Jindal",
```

***Important :** Following are the points needs to be taken into consideration while using PUT REQUEST :*

- *We have to pass **id** as path parameter to specify the record to be updated.*
- *We have to pass the complete data along with the required data to be updated while hitting the PUT request, else only the required data to be updated will be set as the new data in association with the **id** used and all the remaining existing data will be lost.*

5. **PATCH REQUEST** using POSTMAN :

Use **URI** as : http://localhost:3000/Batch1\2

Here,

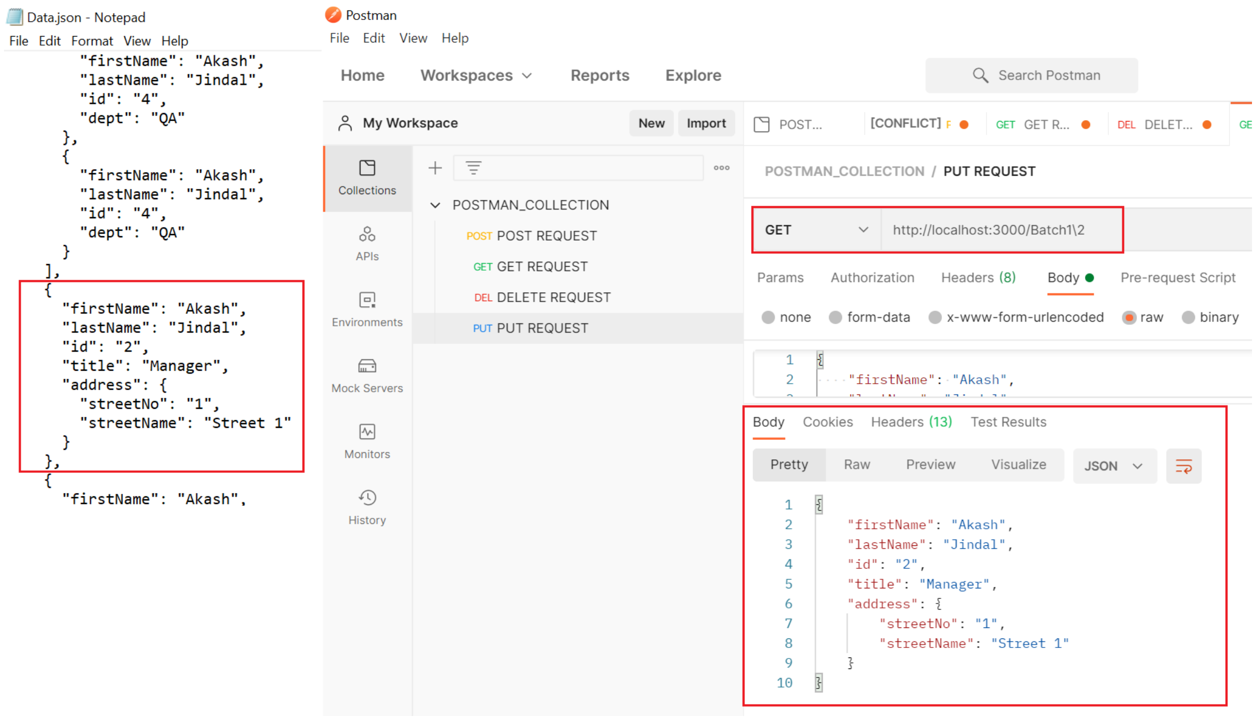
URI : Base + Endpoint + Parameter

Base : http://localhost:3000

Endpoint : /Batch1

Path Parameter : /2

First, have a look at the existing data associated with **id : 2** under **Data.json** file and **POSTMAN** :


Data.json - Notepad
File Edit Format View Help
Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
My Workspace New Import
Collections
APIs
Environments
Mock Servers
Monitors
History
POSTMAN_COLLECTION
POST POST REQUEST
GET GET REQUEST
DEL DELETE REQUEST
PUT PUT REQUEST
POST... [CONFLICT] GET GET R... DEL DELET...
POSTMAN_COLLECTION / PUT REQUEST
Params Authorization Headers (8) Body Pre-request Script
none form-data x-www-form-urlencoded raw binary
Body Cookies Headers (13) Test Results
Pretty Raw Preview Visualize JSON

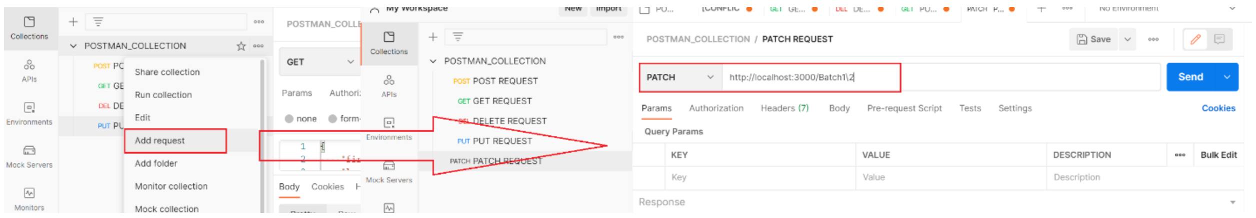
```
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      },
      {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      }
    ],
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "2",
      "title": "Manager",
      "address": {
        "streetNo": "1",
        "streetName": "Street 1"
      }
    },
    {
      "firstName": "Akash",
```

GET http://localhost:3000/Batch1\2

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    }
}
```

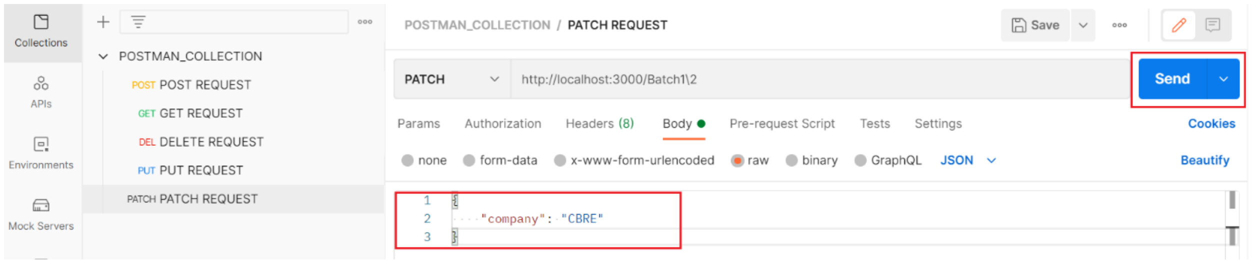
Create a new request under COLLECTIONS.

Select **PATCH request from the HTTP Requests list** and paste the required API.



PATCH http://localhost:3000/Batch1\2

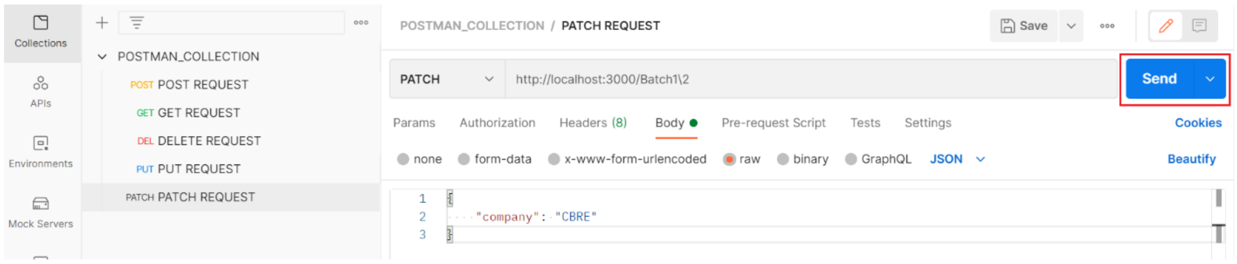
Update the required data : **from : title : Manager to : company : CBRE** :



PATCH http://localhost:3000/Batch1\2

```
{
    "company": "CBRE"
}
```

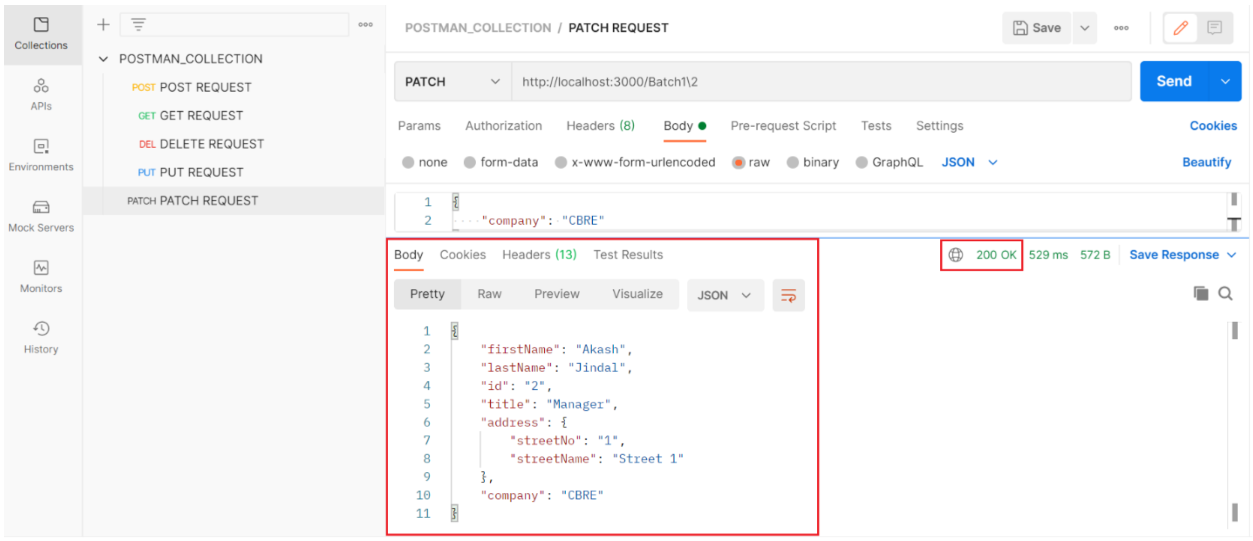
Click **Send.**



PATCH http://localhost:3000/Batch1\2

```
{
    "company": "CBRE"
}
```

Validate the **Status Code** and the received data :



PATCH http://localhost:3000/Batch1\2

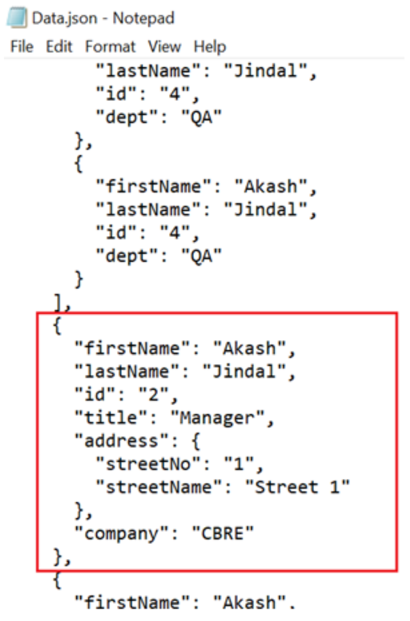
```
{
    "company": "CBRE"
```

200 OK 529 ms 572 B

```
{
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
        "streetNo": "1",
        "streetName": "Street 1"
    },
    "company": "CBRE"
}
```

Executing the command will update the record having **id : 2**.

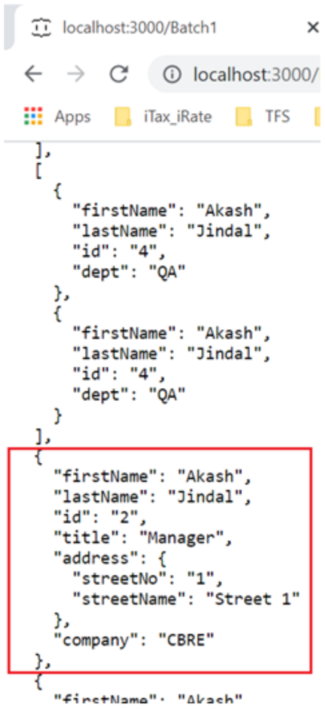
Validate the same under the **Data.json** file :



```
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      },
      {
        "firstName": "Akash",
        "lastName": "Jindal",
        "id": "4",
        "dept": "QA"
      }
    ],
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "2",
      "title": "Manager",
      "address": {
        "streetNo": "1",
        "streetName": "Street 1"
      },
      "company": "CBRE"
    },
    {
      "firstName": "Akash".
```

Validate the same using browser's tab :

Make sure to use the same API used for PATCH Request :



```
  ],
  [
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    },
    {
      "firstName": "Akash",
      "lastName": "Jindal",
      "id": "4",
      "dept": "QA"
    }
  ],
  {
    "firstName": "Akash",
    "lastName": "Jindal",
    "id": "2",
    "title": "Manager",
    "address": {
      "streetNo": "1",
      "streetName": "Street 1"
    },
    "company": "CBRE"
  },
  {
```

***Important :** Following are the points needs to be taken into consideration while using PATCH REQUEST :*

- *We have to pass **id** as path parameter to specify the record to be updated.*
- *We can pass only the required data to be added or modified while hitting the PATCH request and no need of passing the complete associated data.*