Node.js - RESTful API

A Node.js application using ExpressJS is ideally suited for building REST APIs. In this chapter, we shall explain what is a REST (also called RESTFul) API, and build a Node.js based, Express.js REST application. We shall also use REST clients to test out REST API.

API is an acronym for Application Programming Interface. The word interface generally refers to a common meeting ground between two isolated and independent environments. A programming interface is an interface between two software applications. The term REST API or RESTFul API is used for a web Application that exposes its resources to other web/mobile applications through the Internet, by defining one or more endpoints which the client apps can visit to perform read/write operations on the host's resources.

REST architecture has become the de facto standard for building APIs, preferred by the developers over other technologies such as RPC, (stands for Remote Procedure Call), and SOAP (stands for Simple Object Access Protocol).

What is REST architecture?

REST stands for REpresentational State Transfer. REST is a well known software architectural style. It defines how the architecture of a web application should behave. It is a resource based architecture where everything that the REST server hosts, (a file, an image, or a row in a table of a database), is a resource, having many representations. REST was first introduced by Roy Fielding in 2000.

REST recommends certain architectural constraints.

- Uniform interface
- Statelessness
- Client-server
- Cacheability
- Layered system
- Code on demand

These are the advantages of REST constraints –

- Scalability
- Simplicity
- Modifiability

- Reliability
- Portability
- Visibility

A REST Server provides access to resources and REST client accesses and modifies the resources using HTTP protocol. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML but JSON is the most popular one.

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

POST Method

The POST verb in the HTTP request indicates that a new resource is to be created on the server. It corresponds to the CREATE operation in the CRUD (CREATE, RETRIEVE, UPDATE and DELETE) term. To create a new resource, you need certain data, it is included in the request as a data header.

Examples of POST request –

```
HTTP POST http://example.com/users
HTTP POST http://example.com/users/123
```

GET Method

The purpose of the GET operation is to retrieve an existing resource on the server and return its XML/JSON representation as the response. It corresponds to the READ part in the CRUD term.

Examples of a GET request –

```
HTTP GET http://example.com/users
HTTP GET http://example.com/users/123
```

PUT Method

The client uses HTTP PUT method to update an existing resource, corresponding to the UPDATE part in CRUD). The data required for update is included in the request body.

Examples of a PUT request –

```
HTTP PUT http://example.com/users/123
HTTP PUT http://example.com/users/123/name/Ravi
```

DELETE Method

The DELETE method (as the name suggest) is used to delete one or more resources on the server. On successful execution, an HTTP response code 200 (OK) is sent.

Examples of a DELETE request –

```
HTTP DELETE http://example.com/users/123
HTTP DELETE http://example.com/users/123/name/Ravi
```


RESTful Web Services

Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, which provides resource representation such as JSON and set of HTTP Methods.

Creating RESTful API for A Library

Consider we have a JSON based database of users having the following users in a file users.json:

```
{
   "user1" : {
      "name" : "mahesh",
      "password" : "password1",
      "profession" : "teacher",
      "id": 1
   },

   "user2" : {
      "name" : "suresh",
      "password" : "password2",
      "profession" : "librarian",
      "id": 2
   },
```

```
   "user3" : {
      "name" : "ramesh",
      "password" : "password3",
      "profession" : "clerk",
      "id": 3
   }
}
```

Our API will expose the following endpoints for the clients to perform CRUD operations on the users.json file, which the collection of resources on the server.

Sr.No.	URI	HTTP Method	POST body	Result
1	/	GET	empty	Show list of all the users.
2	/	POST	JSON String	Add details of new user.
3	/:id	DELETE	JSON String	Delete an existing user.
4	/:id	GET	empty	Show details of a user.
5	/:id	PUT	JSON String	Update an existing user

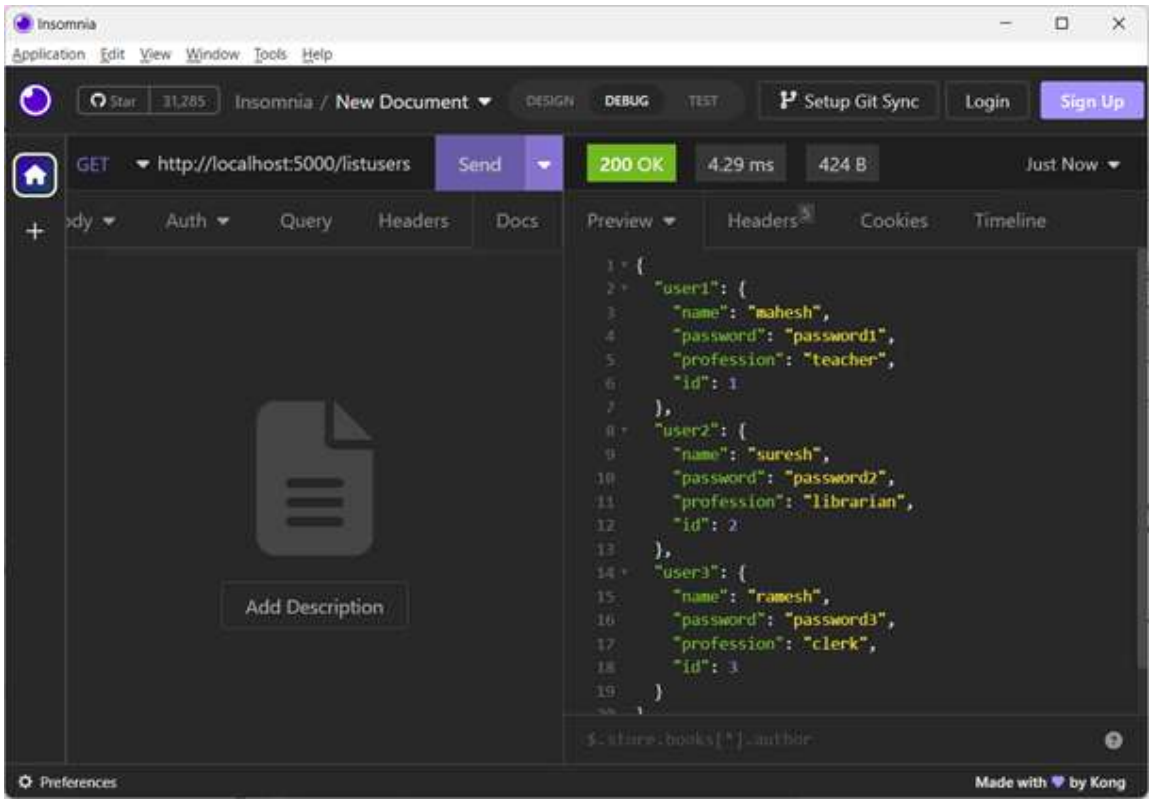
List Users

Let's implement the first route in our RESTful API to list all Users using the following code in a index.js file

```
var express = require('express');
var app = express();
var fs = require("fs");
app.get('/', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      res.end( data );
   });
})
var server = app.listen(5000, function () {
   console.log("Express App running at http://127.0.0.1:5000/");
})
```

To test this endpoint, you can use a REST client such as Postman or Insomnia. In this chapter, we shall use Insomnia client.

Run index.js from command prompt, and launch the Insomnia client. Choose GET methos and enter http://localhost:5000/ URL. The list of all users from users.json will be displayed in the Respone Panel on right.



You can also use CuRL command line tool for sending HTTP requests. Open another terminal and issue a GET request for the above URL.

```
C:\Users\mlath>curl http://localhost:5000/
{
   "user1" : {
      "name" : "mahesh",
      "password" : "password1",
      "profession" : "teacher",
      "id": 1
   },

   "user2" : {
      "name" : "suresh",
      "password" : "password2",
      "profession" : "librarian",
      "id": 2
   },

   "user3" : {
```

```
      "name" : "ramesh",
      "password" : "password3",
      "profession" : "clerk",
      "id": 3
   }
}
```

Show Detail

Now we will implement an API endpoint /:id which will be called using user ID and it will display the detail of the corresponding user.

Add the following method in index.js file –

```
app.get('/:id', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      var users = JSON.parse( data );
      var user = users["user" + req.params.id]
      res.end( JSON.stringify(user));
   });
})
```

In the Insomnia interface, enter http://localhost:5000/2 and send the request.

You may also use the CuRL command as follows to display the details of user2 –

```
C:\Users\mlath>curl http://localhost:5000/2
{"name":"suresh","password":"password2","profession":"librarian","id":2}
```

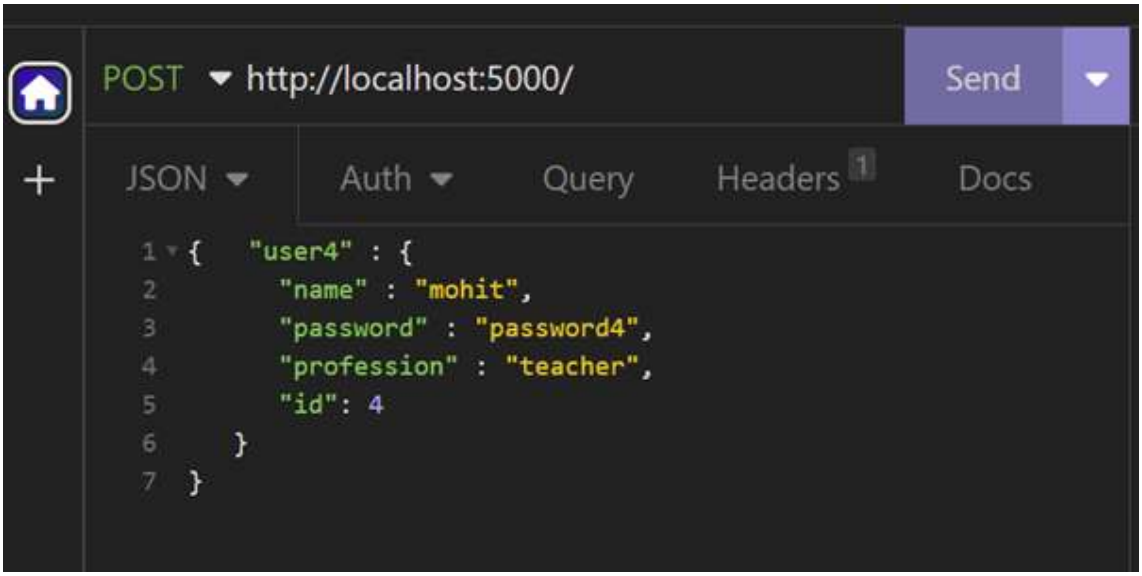
Add User

Following API will show you how to add new user in the list. Following is the detail of the new user. As explained earlier, you must have installed body-parser package in your application folder.

```
var bodyParser = require('body-parser')
app.use( bodyParser.json() );
app.use(bodyParser.urlencoded({  extended: true }));
```

```
app.post('/', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      var users = JSON.parse( data );
      var user = req.body.user4;
      users["user"+user.id] = user
      res.end( JSON.stringify(users));
   });
})
```

To send POST request through Insomnia, set the BODY tab to JSON, and enter the the user data in JSON format as shown



You will get a JSON data of four users (three read from the file, and one added)

```
{
    "user1": {
        "name": "mahesh",
        "password": "password1",
        "profession": "teacher",
        "id": 1
    },
    "user2": {
        "name": "suresh",
        "password": "password2",
        "profession": "librarian",
        "id": 2
    },
    "user3": {
```

```
        "name": "ramesh",
        "password": "password3",
        "profession": "clerk",
        "id": 3
    },
    "user4": {
        "name": "mohit",
        "password": "password4",
        "profession": "teacher",
    "id": 4
    }
  }
```

Delete user

The following function reads the ID parameter from the URL, locates the user from the list that is obtained by reading the users.json file, and the corresponding user is deleted.

```
app.delete('/:id', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      data = JSON.parse( data );
      var id = "user"+req.params.id;
      var user = data[id];
      delete data[ "user"+req.params.id];
      res.end( JSON.stringify(data));
   });
})
```

Choose DELETE request in Insomnia, enter http://localhost:5000/2 and send the request. The user with ID=3 will be deleted, the remaining users are listed in the response panel

Output

```
{
    "user1": {
        "name": "mahesh",
```

```
        "password": "password1",
        "profession": "teacher",
        "id": 1
    },
    "user2": {
        "name": "suresh",
        "password": "password2",
        "profession": "librarian",
        "id": 2
    }
}
```

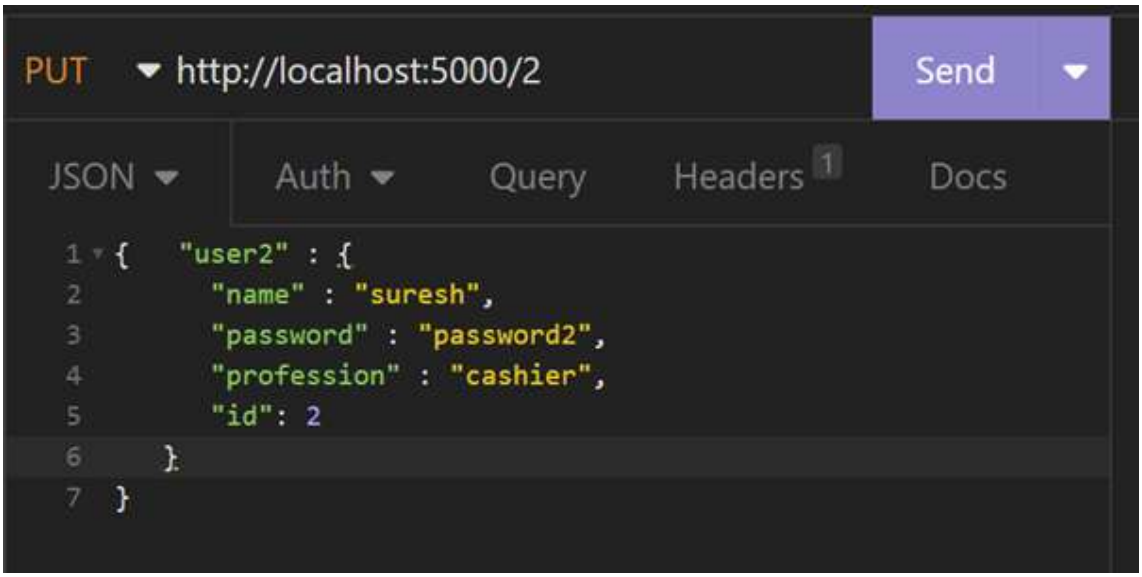
Update user

The PUT method modifies an existing resource with the server. The following app.put() method reads the ID of the user to be updated from the URL, and the new data from the JSON body.

```
app.put("/:id", function(req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {

      var users = JSON.parse( data );
      var id = "user"+req.params.id;
      users[id]=req.body;
      res.end( JSON.stringify(users));
   })
})
```

In Insomnia, set the PUT method for http://localhost:5000/2 URL.



The response shows the updated details of user with ID=2

```
{
    "user1": {
        "name": "mahesh",
        "password": "password1",
        "profession": "teacher",
    "id": 1
    },
    "user2": {
        "name": "suresh",
        "password": "password2",
        "profession": "Cashier",
        "id": 2
    },
    "user3": {
        "name": "ramesh",
        "password": "password3",
        "profession": "clerk",
        "id": 3
    }
}
```

Here is the complete code for the Node.js RESTFul API –

```
var express = require('express');
var app = express();
var fs = require("fs");
```

```
var bodyParser = require('body-parser')
app.use( bodyParser.json() );
app.use(bodyParser.urlencoded({  extended: true }));


app.get('/', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      res.end( data );
   });
})

app.get('/:id', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      var users = JSON.parse( data );
      var user = users["user" + req.params.id]
     res.end( JSON.stringify(user));
   });
})

var bodyParser = require('body-parser')
app.use( bodyParser.json() );
app.use(bodyParser.urlencoded({  extended: true }));

app.post('/', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      var users = JSON.parse( data );
      var user = req.body.user4;
      users["user"+user.id] = user
      res.end( JSON.stringify(users));
   });
})

app.delete('/:id', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
      data = JSON.parse( data );
      var id = "user"+req.params.id;
      var user = data[id];
      delete data[ "user"+req.params.id];
      res.end( JSON.stringify(data));
   });
})
app.put("/:id", function(req, res) {
```

```
      fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err,
data) {

      var users = JSON.parse( data );
      var id = "user"+req.params.id;

      users[id]=req.body;
      res.end( JSON.stringify(users));
   })

})
var server = app.listen(5000, function () {
   console.log("Express App running at http://127.0.0.1:5000/");
})
```