

IP PRACTICAL NO. 12

Date of Performance: 25 – 10 – 2022

Software Requirement: Visual Studio Code, NodeJS, ExpressJS, Thunder Client.

Aim: To Implement REST API in Express JS.

Objectives: The aim of this experiment is that the students will be able to:

- Implement REST API methods.
- Understand different features of REST API & ExpressJS .

Outcomes: After study of this experiment, the students will be able to:

- Create server-side codes and implement API functionalities.
- Implement the usage of REST API in Express JS.

Prerequisite: Basic knowledge of JavaScript, ExpressJS and API.

Theory:

Express.js, or simply Express, is a back end web application framework for building RESTful APIs with Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js. Core features of Express framework:

- It can be used to design single-page, multi-page and hybrid web applications.
- It allows to setup middleware to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- It allows to dynamically render HTML Pages based on passing arguments to templates.

REST(Representational state transfer) **API**(Application Programming Interface) is the standard way to send and receive data for web services. A client sends a request which first goes to the rest API and then to the database to get or put the data after that, it will again go to the rest API and then to the client. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server.

HTTP Request Types

HTTP Requests are simply messages that are sent by the client to do some tasks on the server.

- GET - Get command is used to request data from the server, but mainly this method is used to read data
- PUT - This command is used to completely replace the resource with the submitted resource, typically used to update data.
- POST - The post method is used to create new or to edit already existing data
- Delete - This delete command is used to delete the data completely from the server

Source Code:

Index.js:

```
const express = require('express');
const routes = require('./routes.js'); // import the routes
const app = express();
app.use(express.json());
app.use('/', routes); //to use the routes
const listener = app.listen(process.env.PORT || 3000, (err) => {
  if (err) {
    console.log(err);
  }
  console.log('Your app is listening on port ' + listener.address().port)
  console.log()
})
```

routes.js:

```
const express = require('express'); //import express
const router = express.Router();
router.get('/mammal', (req, res) => {
  res.json({message: "GET all mammal"});
});
router.post('/mammal', (req, res) => {
  res.json({message: "Add New mammal to DB with POST"});
});
router.delete('/mammal', (req, res) => {
  res.json({message: "Delete a mammal from DB"});
});
router.put('/mammal', (req, res) => {
  res.json({message: "Add New mammal to DB with PUT"});
});
router.get('/mammal/:name', (req, res) => {
  res.json({ message: "Get mammal of name : " + req.params.name });
});
router.post('/mammal/:name', (req, res) => {
  res.json({message: "POST mammal of name : " + req.params.name});
});
router.delete('/mammal/:name', (req, res) => {
  res.json({message: "DELETE mammal of name : " + req.params.name});
});
module.exports = router; // export to use in server.js
```

Output:

```
PS C:\Users\husna\OneDrive\Desktop\Reactprac\exp12trial> node index.js
Your app is listening on port 3000
```

GET '/mammal' Method :-

The screenshot shows a web client interface with a 'Query' tab selected. The URL bar shows 'http://localhost:3000/mammal' and a 'Send' button. Below the URL bar, there are tabs for 'Query', 'Headers', 'Auth', 'Body', 'Tests', and 'Pre Run'. The 'Query' tab is active, showing 'Query Parameters' with a table for parameters. The 'Response' tab is also visible, showing the response body:

```
{
  "message": "GET all mammal"
}
```

 The status bar at the top right indicates 'Status: 200 OK', 'Size: 28 Bytes', and 'Time: 30 ms'.

POST '/mammal' Method :-

POST	http://localhost:3000/mammal	Send	Status: 200 OK Size: 44 Bytes Time: 5 ms									
Query	Headers 2	Auth	Body	Tests	Pre Run New	Response	Headers 6	Cookies	Results	Docs	{}	≡
Query Parameters						1 { 2 "message": "Add New mammal to DB with POST" 3 }						
<input type="checkbox"/> parameter value												

PUT '/mammal' Method :-

PUT	http://localhost:3000/mammal	Send	Status: 200 OK Size: 43 Bytes Time: 10 ms									
Query	Headers 2	Auth	Body	Tests	Pre Run New	Response	Headers 6	Cookies	Results	Docs	{}	≡
Query Parameters						1 { 2 "message": "Add New mammal to DB with PUT" 3 }						
<input type="checkbox"/> parameter value												

DELETE '/mammal' Method :-

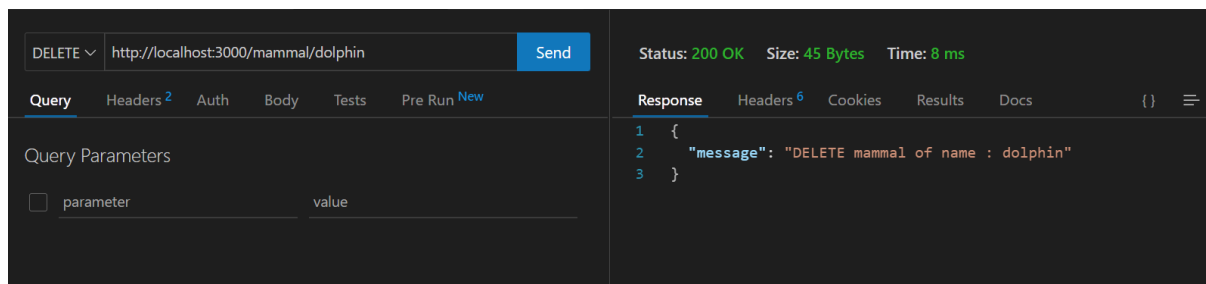
DELETE	http://localhost:3000/mammal	Send	Status: 200 OK Size: 37 Bytes Time: 6 ms									
Query	Headers 2	Auth	Body	Tests	Pre Run New	Response	Headers 6	Cookies	Results	Docs	{}	≡
Query Parameters						1 { 2 "message": "Delete a mammal from DB" 3 }						
<input type="checkbox"/> parameter value												

GET '/mammal/:name' Method :-

GET	http://localhost:3000/mammal/dolphin	Send	Status: 200 OK Size: 42 Bytes Time: 6 ms									
Query	Headers 2	Auth	Body	Tests	Pre Run New	Response	Headers 6	Cookies	Results	Docs	{}	≡
Query Parameters						1 { 2 "message": "Get mammal of name : dolphin" 3 }						
<input type="checkbox"/> parameter value												

POST '/mammal/:name' Method :-

POST	http://localhost:3000/mammal/dolphin	Send	Status: 200 OK Size: 43 Bytes Time: 5 ms									
Query	Headers 2	Auth	Body	Tests	Pre Run New	Response	Headers 6	Cookies	Results	Docs	{}	≡
Query Parameters						1 { 2 "message": "POST mammal of name : dolphin" 3 }						
<input type="checkbox"/> parameter value												

DELETE '/mammal/:name' Method :-

The screenshot displays a REST client interface. The top bar shows the HTTP method 'DELETE' and the URL 'http://localhost:3000/mammal/dolphin'. The 'Send' button is visible. Below the top bar, the 'Query' tab is selected, showing 'Query Parameters' with a table for parameters. The 'Response' tab is also visible, showing the response status '200 OK', size '45 Bytes', and time '8 ms'. The response body is a JSON object:

```
{  "message": "DELETE mammal of name : dolphin"}
```

parameter	value
-----------	-------

```
1 {
2   "message": "DELETE mammal of name : dolphin"
3 }
```

Conclusion:

This experiment has touched on topics, what is rest API, and HTTP request types, and we also have created a simple Rest API using Express.js. We have learned the importance of these technologies and how to implement them.