COURSEWARE

REST applications with Node

Contents

Overview

In the module, we will learn how to create a simple Node application serving REST API.

A REST API (also referred to as RESTful API) is an application programming interface (API) that follows the rules of REST architectural style. The six rules of the REST architecture are:

- Client–server architecture
- Statelessness
- Cacheability
- Layered system
- Code on demand (optional)
- Uniform interface

In practice, REST APIs are HTTP endpoints exposing database entities as resources. Such resources can be accessed and manipulated using REST verbs such as:

- GET
- POST
- PUT
- DELETE

Tutorial

In this tutorial, we will create a simple REST application in Node. First of all, let's create a new directory for our application:

```
mkdir rest-api
```

Now change your current directory to the one we have just created:

```
cd rest-api
```

The next step is to initialize an empty Node application:

```
$ npm init -y
Wrote to /tmp/rest-api/package.json:

{
"name": "rest-api",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC"
}
```

Then we would like to create a file named `app.js` and add the following code there:

```
const express = require('express')
const app = express()
const port = 8080

// Add URL handler
app.get('/', (req, res) => {
    res.end('Hello!');
});

// Add HTTP listener
app.listen(port, () => {
    console.log(`REST API application listening at http://localhost:${port}`)
});
```

This is a very simple application that uses Express to serve REST API over HTTP TCP endpoint. Express is one of the most popular REST API frameworks for Node. As you can see Express can be used to create a working REST API application with just a couple of lines of code.

We register URL handlers to our express application. Our handler is registered to `/` URL and simply returns `Hello!` as an HTTP response. Then we start our Express application TCP endpoint listener on TCP port 8080.

Before we start testing our application, we have to install Express dependency:

```
$ npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN rest-api@1.0.0 No description
npm WARN rest-api@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 4.198s
found 0 vulnerabilities
```

Now we can start our application using the following command:

```
$ node app.js
REST API application listening at http://localhost:8080
```

Let's test our new application using an HTTP client like curl:

```
$ curl localhost:8080
Hello!
```

Yay! It worked - we can see an expected HTTP response.

Adding JSON support

Now let's make our example a bit more complicated. We would like to return JSON response as a real REST API does.

In order to do this, we will use the `JSON.stringify` function to generate a JSON response. We will create a mock product object representing an entity from a mock e-commerce system.

```
const product = { title: "The Office DVD", price: 25.99 };
const json = JSON.stringify(product);
```

We would like to replace our static `Hello!` response with our JSON response instead:

```
const express = require('express')
const app = express()
const port = 8080

app.get('/', (req, res) => {
    const product = { title: "The Office DVD", price: 25.99 };
    const json = JSON.stringify(product);
    res.end(json);
});

app.listen(port, () => {
    console.log(`REST API application listening at http://localhost:${port}`)
});
```

Let's give it a try!

```
$ curl localhost:8080
{"title":"The Office DVD","price":25.99}
```

The response seems to be what we expected! However, if we dig into it a little more with curl `-v` (verbose) option we can see that it doesn't return the JSON response type header.

```
$ curl -v localhost:8080
*   Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Date: Tue, 10 Aug 2021 08:51:06 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Content-Length: 40
<
* Connection #0 to host localhost left intact
{"title":"The Office DVD","price":25.99}
```

As REST APIs should indicate what an encoding they are returning, we should add `res.setHeader('Content-Type', 'application/json');` line to enhance our response with the information about a returned data format:

```
const express = require('express')
const app = express()
const port = 8080

app.get('/', (req, res) => {
    const product = { title: "The Office DVD", price: 25.99 };
    const json = JSON.stringify(product);
    res.setHeader('Content-Type', 'application/json');
    res.end(json);
});

app.listen(port, () => {
    console.log(`REST API application listening at http://localhost:${port}`)
});
```

Let's test it again with `curl -v`:

```
$ curl -v localhost:8080
*   Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: application/json
< Date: Tue, 10 Aug 2021 08:52:01 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Content-Length: 40
<
* Connection #0 to host localhost left intact
{"title":"The Office DVD","price":25.99}
```

It's much better now!

Exercises

There are no exercises for this module.