Calling and Developing RESTful APIs Using Node.js

Assoc Prof. Dr. Kanda Runapongsa Saikaew
Department of Computer Engineering
Khon Kaen University
krunapon@kku.ac.th

Agenda

- **What is REST?**
- Calling REST API
- Developing REST API

What is REST?

- REST is an acronym for Representational State Transfer
- It is web standards architecture and HTTP protocol
- The REST architectural style describes six constraints that were originally communicated by Roy Fielding

The basis of RESTful-style

1. Uniform Interface
2. Stateless
3. Cacheable
4. Client-Server
5. Layered System
6. Code on Demand (optional)

HTTP requests

- RESTful applications use HTTP requests to perform four operations termed as CRUD (C:create, R:read, U:update, and D:delete)
- Create and/or update is used to post data, get for reading/listing data, and delete to remove data
- RESTful is composed of methods such as: based URL, URL, media types

Agenda

- What is REST?
- **Calling REST API**
- Developing REST API

A remote web service to test

- To save us the trouble of first building a remote web service, we're going to make use of a very popular and very easy to use web service offered by httpbin
- With this service we can issue requests and essentially get what we've sent, back as a response.

Installing NPM request

- The first step is to include the appropriate Node.js package in the project
- Using the Node Package Manager (NPM), execute the following
  - npm install request --save
- The above command will download the request package and save it to our **package.json** file

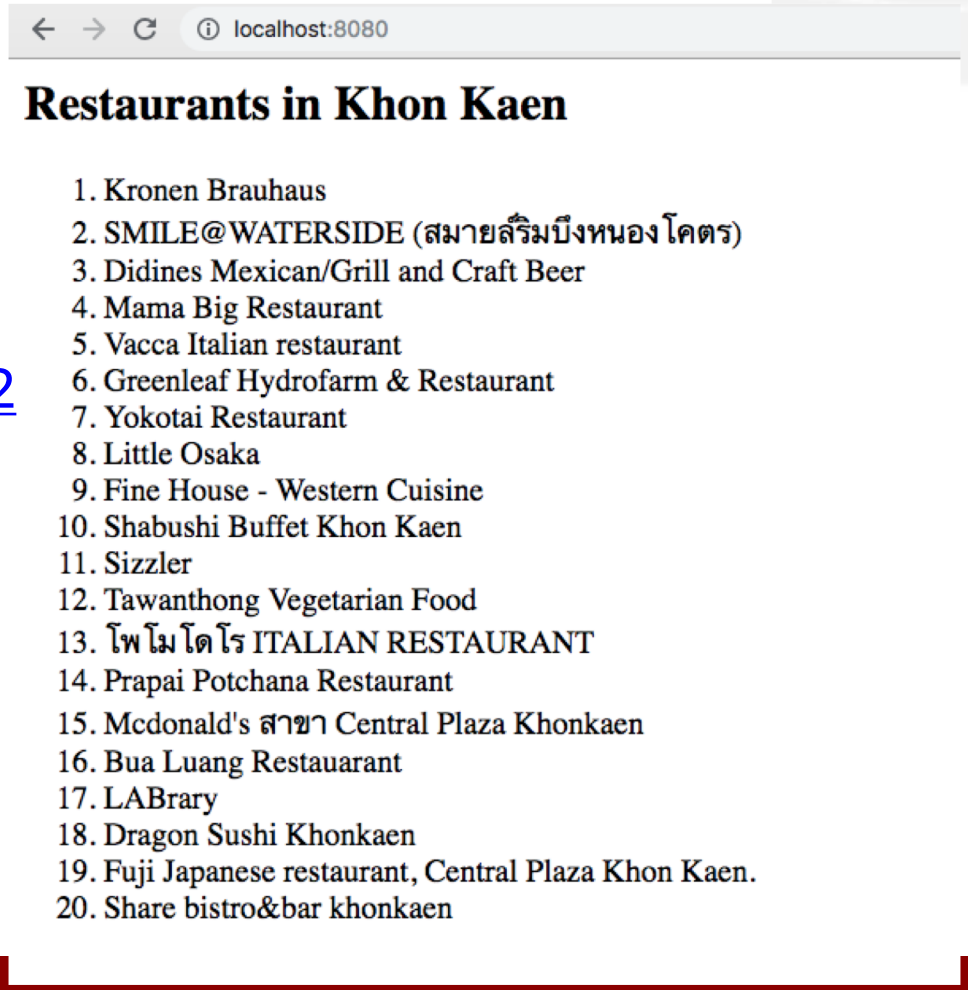
Sample Get Request

```
1  var request1 = require("request");
2  request1.get("http://httpbin.org/ip", (error, response, body) => {
3      if (error) {
4          return console.dir(error);
5      }
6      var result = JSON.parse(body);
7      var ip = result.origin;
8      console.dir(ip);
9  });
```

```
[~/Dropbox/Sites/nodejs]$ node request1.js
'223.24.165.178'
[~/Dropbox/Sites/nodejs]$
```

Exercise 1: Calling Google Maps

```
[~/Dropbox/Sites/nodejs]$ node call-google-map.js
```



https://maps.googleapis.com/maps/api/place/textsearch/json?query=restaurants%20in%20Khon%20Kaen&key=<key>

Sample Post Request

```
1  var request1 = require("request");
2
3  request1.post({
4      "headers": { "content-type": "application/json" },
5      "url": "http://httpbin.org/post",
6      "body": JSON.stringify({
7          "firstname": "Manee",
8          "lastname": "Jaidee"
9      })
10 }, (error, response, body) => {
11     if(error) {
12         return console.dir(error);
13     }
14     console.dir(JSON.parse(body));
15 });
```

```
[~/Dropbox/Sites/nodejs]$  node post1.js
{ args: {},
  data: '{"firstname":"Manee","lastname":"Jaidee"}',
  files: {},
  form: {},
  headers:
   { Connection: 'close',
     'Content-Length': '41',
     'Content-Type': 'application/json',
     Host: 'httpbin.org' },
  json: { firstname: 'Manee', lastname: 'Jaidee' },
  origin: '223.24.165.178',
  url: 'http://httpbin.org/post' }
```

Tools

- Node.js
- MongoDB
- Text editor
- Postman

Getting started

- For the purpose of this tutorial, I'll work you through creating a RESTful API
- To achieve this, we will create a RESTful todo list API
  - Endpoints that will create a task, get or read list of all tasks, read a particular task, delete a task, and update a task

Assumptions

- Presume that you already have your environment set up (i.e Node.js and mongoDB is installed)
- Kindly run npm –v and mongo –version as these will show you the version of NPM and MongoDB installed on your machine

```
[~/Dropbox/Sites/nodejs/api]$  npm -v
6.4.1
[~/Dropbox/Sites/nodejs/api]$  mongo --version
MongoDB shell version v4.0.0
git version: 3b07af3d4f471ae89e8186d33bbb1d5259597d51
allocator: system
modules: none
build environment:
    distarch: x86_64
    target_arch: x86_64
```

Performing these steps (1/5)

1. Create a folder name todoListApi

    - mkdir todoListApi

2. Navigate to the root of your newly created folder

    - cd todoListApi

```
[~/Dropbox/Sites/nodejs/api]$  mkdir todoListApi
[~/Dropbox/Sites/nodejs/api]$  cd todoListApi/
 [~/Dropbox/Sites/nodejs/api/todoListApi]$
```

Performing these steps (2/5)

3. Create a package.json file – npm init

```
[~/Dropbox/Sites/nodejs/api/todoListApi]$  npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (todolistapi)
version: (1.0.0)
description:
entry point: (index.js)
```

Performing these steps (3/5)

4. Create a file called server.js

    - touch server.js

5. Create a folder called api

    - mkdir api

Inside this folder called api, create three separate folders caleld models, routes, and controllers by running

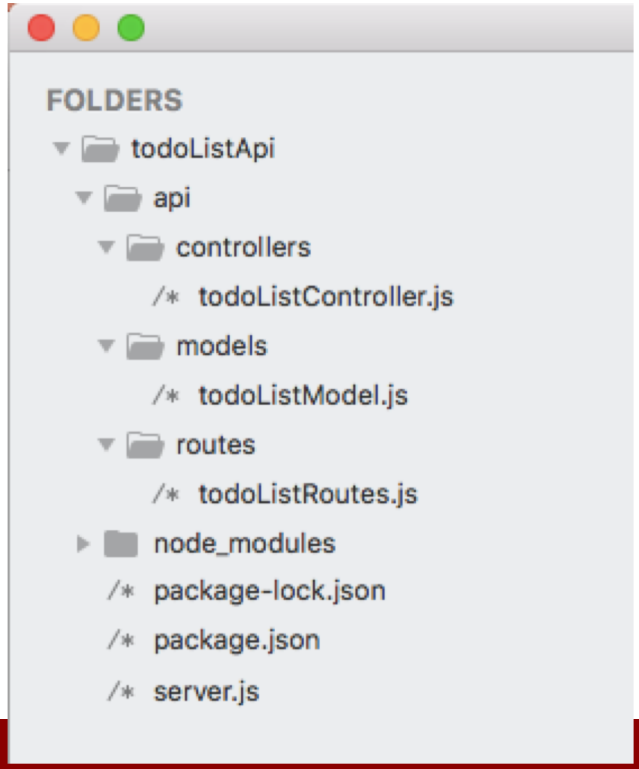
mkdir api/controllers api/modules api/routes

Performing these steps (4/5)

6. Create todoListController.js in the api/controller folder, todoListRoutes.js in the routes folder, and todoListModel in the model folder

- touch api/controllers/todoListController.js
  api/modules/todoListModel.js
  api/routes/todoListRoutes.js

Our folder structure should look like this now.



Server setup

- Let’s install express and nodemon, express will be used to create the server while nodemon will help us to keep track of changes to our application by watching changed files and automatically restart the server

```
npm install --save-dev nodemon
npm install express --save
```

What is Nodemon?

- Nodemon is a utility that will monitor for any changes in your source and automatically restart your server
- Perfect for development
- Install it using npm.
- Just use nodemon instead of node to run your code, and now your process will automatically restart when your code changes.

What is Express?

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications
- Express provides a thin layer of fundamental web application features
- With HTTP utility methods, creating a robust API is quick and easy

Checking whether installation is successful

- On successful installation, your package.json file will be modified to have the two newly installed packages

Start the server (1/3)

1. Open the package.json file and add this task to the script “start”:”nodemon server.js”

Start the server (2/3)

2. Open the server.js file and type/copy the code below into it

todoListApi/server.js (api) — Brackets

```
1  var express = require('express'),
2      app = express(),
3      port = process.env.PORT || 3000;
4
5  app.listen(port);
6  console.log('todo list RESTful API server started on:' + port);
```

Start the server (3/3)

- On your terminal, run npm run start this will start the server and then you will see

```
                                                                                t
[~/Dropbox/Sites/nodejs/api/todoListApi/api/routes]$  npm run start
> todolistapi@1.0.0 start /Users/krunapon/Dropbox/Sites/nodejs/api/todoListApi
> nodemon server.js

[nodemon] 1.18.6
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
todo list RESTful API server started on:3000
```

What is MongoDB? (1/2)

- MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need
- MongoDB **stores data in flexible, JSON-like documents**, meaning fields can vary from document to document and data structure can be changed over time
- The document model **maps to the objects in your application code**, making data easy to work with

What is MongoDB? (2/2)

- **Ad hoc queries, indexing, and real time aggregation** provide powerful ways to access and analyze your data
- MongoDB is a **distributed database at its core**, so high availability, horizontal scaling, and geographic distribution are built in and easy to use
- MongoDB is **free and open-source**, published under the GNU Affero General Public License

Installing Mongoose

- Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment
- After you install node.js and mongodb, then
- npm install mongoose --save

```
[~/Dropbox/Sites/nodejs/api/todoListApi/api/routes]$ npm install mongoose --save
npm WARN todolistapi@1.0.0 No description
npm WARN todolistapi@1.0.0 No repository field.

+ mongoose@5.3.10
updated 1 package and audited 2412 packages in 9.569s
found 0 vulnerabilities
```

Why Mongoose?

- Moongoose is what we will use to interact with a MongoDB (Database) instance
- After installation, open the todoListModel.js file in your api/models folder and type the following code into the file and save

Edit todoListModel.js

- Edit api/models/todoListModel.js

todoListApi/api/modules/todoListModel.js (api) — Brackets

```
1  'use strict';
2  var mongoose = require('mongoose');
3  var Schema = mongoose.Schema;
4
5
6  var TaskSchema = new Schema({
7    name: {
8      type: String,
9      required: 'Kindly enter the name of the task'
10   },
11   Created_date: {
12     type: Date,
13     default: Date.now
14   },
15   status: {
16     type: [{
17       type: String,
18       enum: ['pending', 'ongoing', 'completed']
19     }],
20     default: ['pending']
21   }
22 });
23
24 module.exports = mongoose.model('Tasks', TaskSchema);
```

Setting up the routes (1/2)

- Routing refers to determining how an application responds to a client request for a specific endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on)
- Each of our routes has different route handler functions, which are executed when the route is matched

Setting up the routes (2/2)

- In the next code, there are two basic routes (‘/tasks’, and ‘/tasks/taskId’) with different methods
- ‘/tasks’ has methods GET and POST while ‘/tasks/taskId’ has GET, PUT, and DELETE

todoListApi/api/routes/todoListRoutes.js (api) — Brackets

```
1  'use strict';
2  module.exports = function(app) {
3    var todoList = require('../controllers/todoListController');
4
5    // todoList Routes
6    app.route('/tasks')
7      .get(todoList.list_all_tasks)
8      .post(todoList.create_a_task);
9
10
11   app.route('/tasks/:taskId')
12     .get(todoList.read_a_task)
13     .put(todoList.update_a_task)
14     .delete(todoList.delete_a_task);
15 };
```

Setting up the controller

- In this controller, we would be writing five (5) different functions namely: list_all_tasks, create_a_task, read_a_task, update_a_task, delete_a_task
- Each of these functions uses different mongoose methods such as find, findById, findOneAndUpdate, save, and remove

api/controllers/todoListController.js (1/2)

todoListApi/api/controllers/todoListController.js (api) — Brackets

```
1  'use strict';
2
3
4  var mongoose = require('mongoose'),
5    Task = mongoose.model('Tasks');
6
7  exports.list_all_tasks = function(req, res) {
8    Task.find({}, function(err, task) {
9      if (err)
10       res.send(err);
11     res.json(task);
12   });
13 };
14
15
16
17
18 exports.create_a_task = function(req, res) {
19   var new_task = new Task(req.body);
20   new_task.save(function(err, task) {
21     if (err)
22       res.send(err);
23     res.json(task);
24   });
25 };
```

api/controllers/todoListController.is (2/2)

todoListApi/api/controllers/todoListController.js (api) — Brackets

```
28 exports.read_a_task = function(req, res) {
29   Task.findById(req.params.taskId, function(err, task) {
30     if (err)
31       res.send(err);
32     res.json(task);
33   });
34 };
35
36
37 exports.update_a_task = function(req, res) {
38   Task.findOneAndUpdate({_id: req.params.taskId}, req.body, {new: true},
     function(err, task) {
39     if (err)
40       res.send(err);
41     res.json(task);
42   });
43 };
44
45
46 exports.delete_a_task = function(req, res) {
47
48
49   Task.remove({
50     _id: req.params.taskId
51   }, function(err, task) {
52     if (err)
53       res.send(err);
54     res.json({ message: 'Task successfully deleted' });
55   });
56 };
```

Putting everything together

- Next, we will connect handlers (controllers), database, the created models, body parser, and the created routes together
- Open the server.js file
  - Connect your database
  - Load the created model – task
  - Install bodyParser and use bodyParser parse incoming request bodies
  - Register our created routes in the sever

File server.js

server.js (api) — Brackets

```
1  var express = require('express'),
2    app = express(),
3    port = process.env.PORT || 3000,
4    mongoose = require('mongoose'),
5    Task = require('./api/models/todoListModel'), //created model loading here
6    bodyParser = require('body-parser');
7
8  // mongoose instance connection url connection
9  mongoose.Promise = global.Promise;
10 mongoose.connect('mongodb://localhost/Tododb');
11
12
13 app.use(bodyParser.urlencoded({ extended: true }));
14 app.use(bodyParser.json());
15
16
17 var routes = require('./api/routes/todoListRoutes'); //importing route
18 routes(app); //register the route
19
20
21 app.listen(port);
22
23
24 console.log('todo list RESTful API server started on: ' + port);
```

Start Mongodb

- Using command mongod

```
[~/Dropbox/Sites/nodejs/todoListApi]$ mongod
2018-11-07T10:08:07.912+0700 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify -
-sslDisabledProtocols 'none'
2018-11-07T10:08:08.038+0700 I CONTROL  [initandlisten] MongoDB starting : pid=1090 port=27017 dbpath=/data/db 64
-bit host=macair.local
2018-11-07T10:08:08.039+0700 I CONTROL  [initandlisten] db version v4.0.0
2018-11-07T10:08:08.039+0700 I CONTROL  [initandlisten] git version: 3b07af3d4f471ae89e8186d33bbb1d5259597d51
2018-11-07T10:08:08.039+0700 I CONTROL  [initandlisten] allocator: system
2018-11-07T10:08:08.039+0700 I CONTROL  [initandlisten] modules: none
2018-11-07T10:08:08.039+0700 I CONTROL  [initandlisten] build environment:
2018-11-07T10:08:08.039+0700 I CONTROL  [initandlisten]     distarch: x86_64
2018-11-07T10:08:08.040+0700 I CONTROL  [initandlisten]     target_arch: x86_64
2018-11-07T10:08:08.040+0700 I CONTROL  [initandlisten] options: {}
2018-11-07T10:08:08.045+0700 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=3584M,session_m
ax=20000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=tru
e,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery
_progress),
2018-11-07T10:08:09.127+0700 I STORAGE  [initandlisten] WiredTiger message [1541560089:127625][1090:0x7fffa77b334
0], txn-recover: Set global recovery timestamp: 0
2018-11-07T10:08:09.355+0700 I RECOVERY [initandlisten] WiredTiger recoveryTimestamp. Ts: Timestamp(0, 0)
2018-11-07T10:08:09.506+0700 I CONTROL  [initandlisten]
```

Start node server

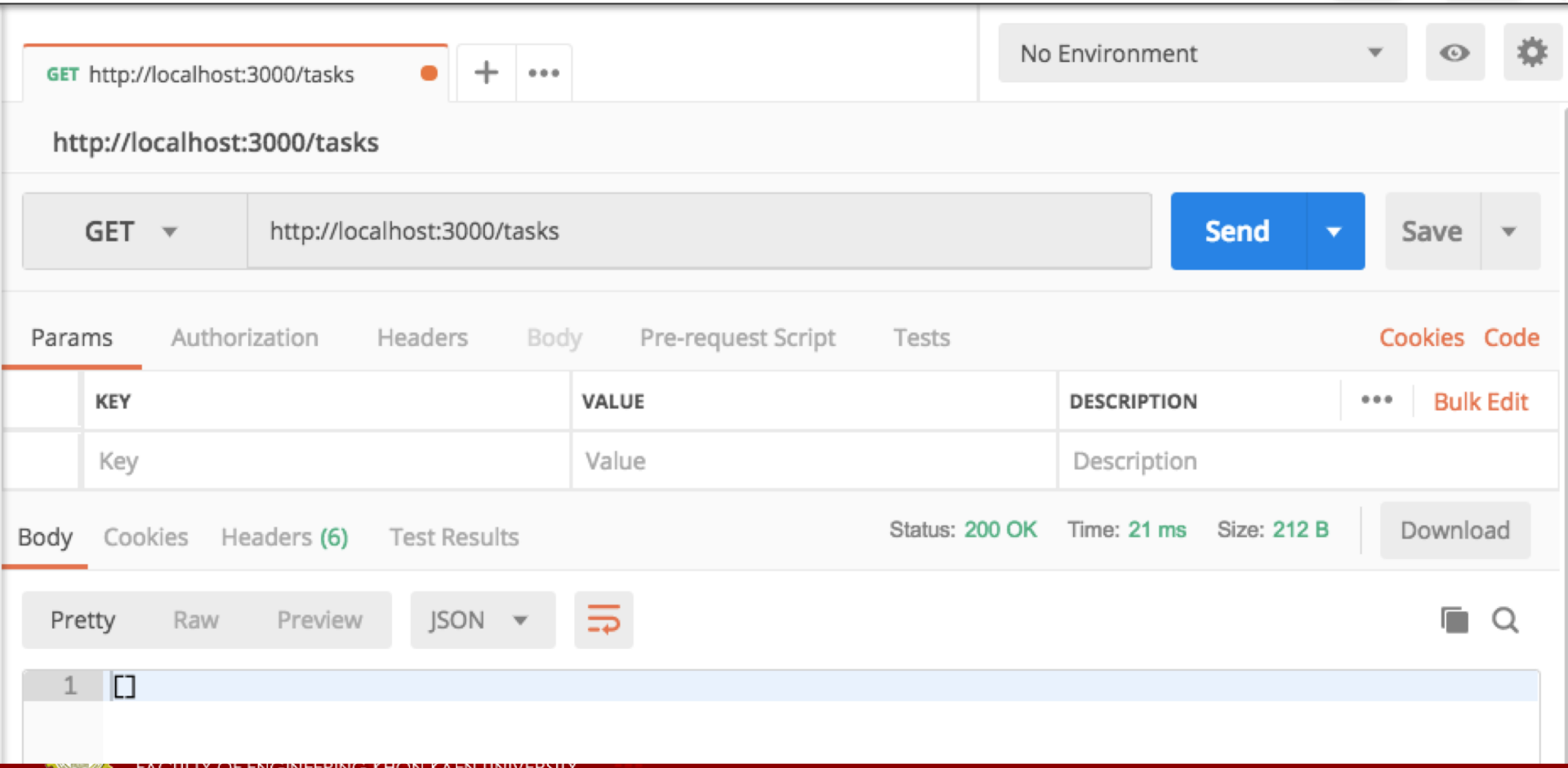
```
[~/Dropbox/Sites/nodejs/todoListApi]$  npm run start

> todolistapi@1.0.0 start /Users/krunapon/Dropbox/Sites/nodejs/todoListApi
> nodemon server.js

[nodemon] 1.18.6
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
todo list RESTful API server started on: 3000
(node:1113) DeprecationWarning: `open()` is deprecated in mongoose >= 4.11.0, us
e `openUri()` instead, or set the `useMongoClient` option if using `connect()` o
r `createConnection()`. See http://mongoosejs.com/docs/4.x/docs/connections.html
#use-mongo-client
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
todo list RESTful API server started on: 3000
(node:1117) DeprecationWarning: `open()` is deprecated in mongoose >= 4.11.0, us
e `openUri()` instead, or set the `useMongoClient` option if using `connect()` o
r `createConnection()`. See http://mongoosejs.com/docs/4.x/docs/connections.html
#use-mongo-client
```

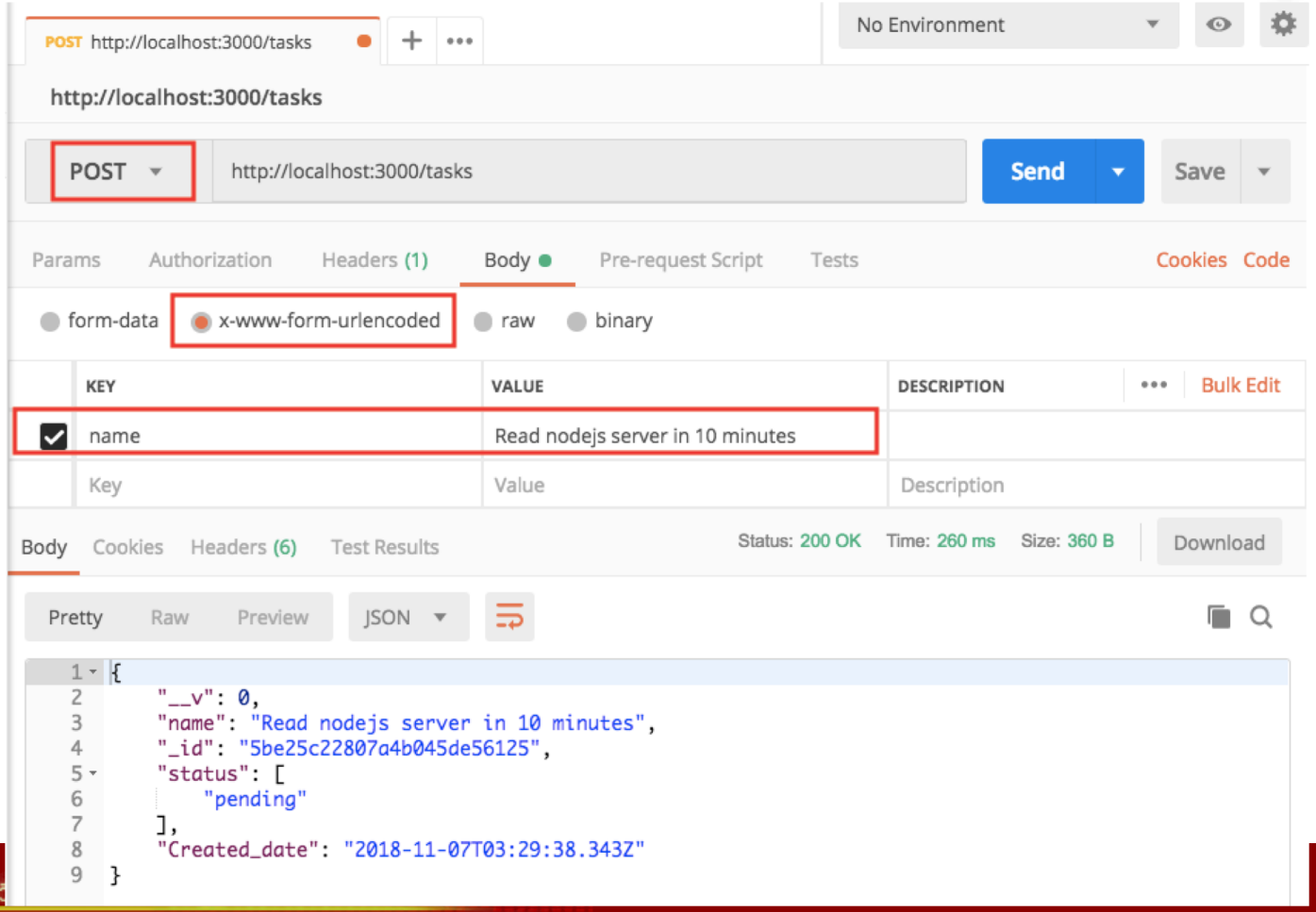
http://localhost:3000/tasks using GET

Open your postman and type http://localhost:3000/tasks in the enter request URL section and press enter.



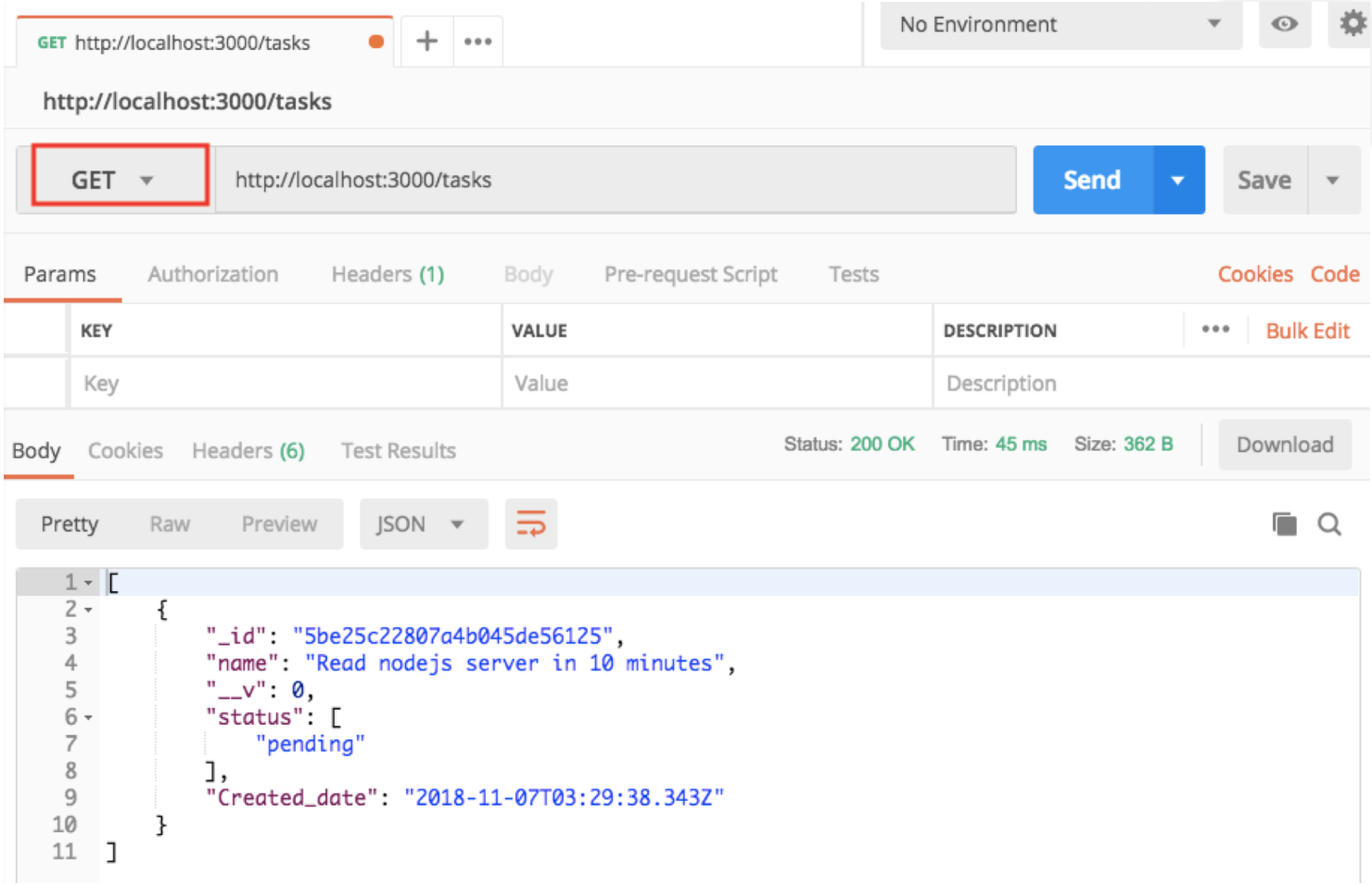
http://localhost:3000/tasks using POST

On the same address, change the method to POST,
click body and select "x-www-form-urlencoded".
Then, enter name as the key and the corresponding task name as value.
After this, click on send button. This should give you a response 200 ok

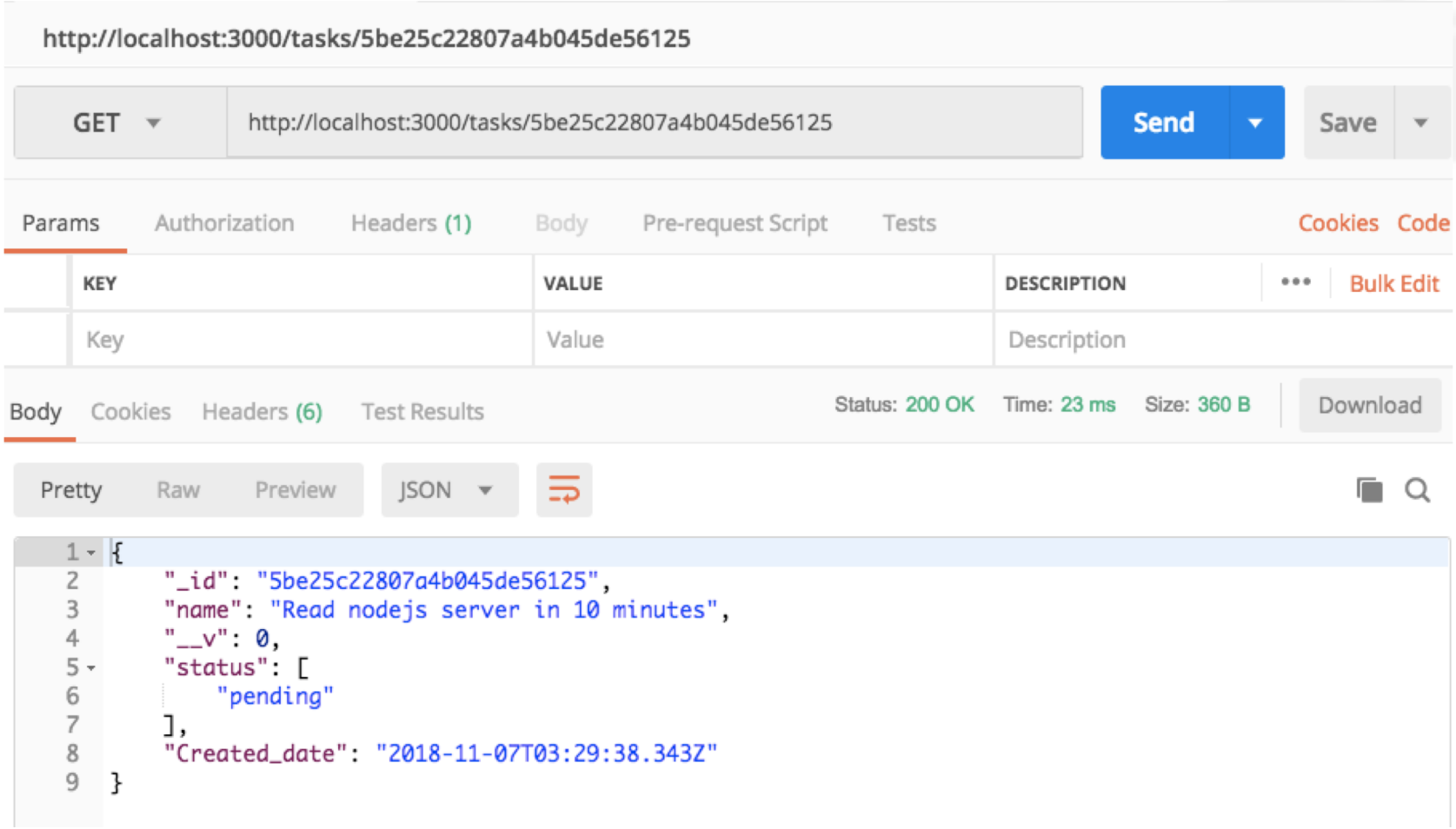


http://localhost:3000/tasks

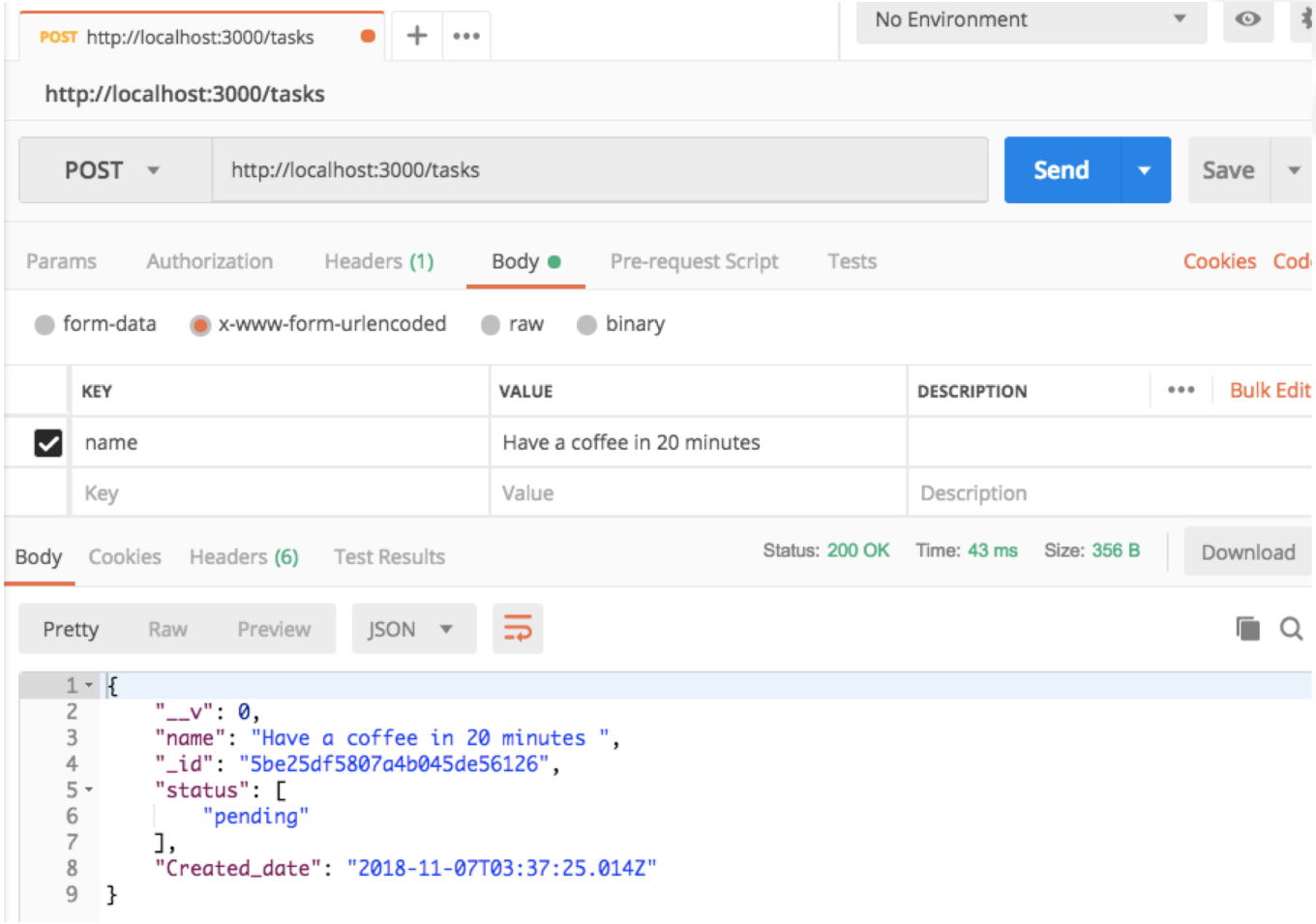
using GET to list all tasks



http://localhost:3000/tasks/<id> using GET

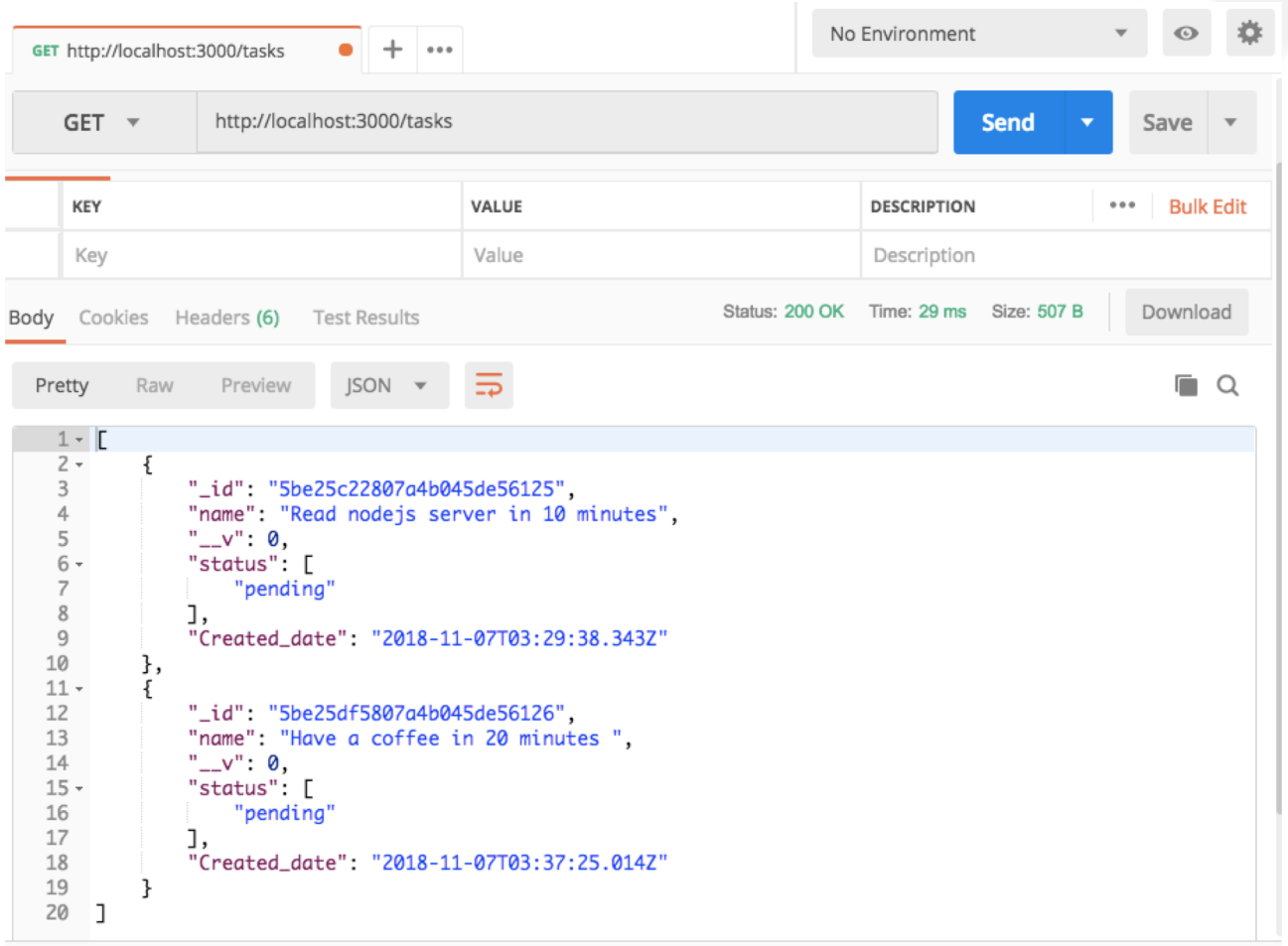


http://localhost:3000/tasks using POST to add another task



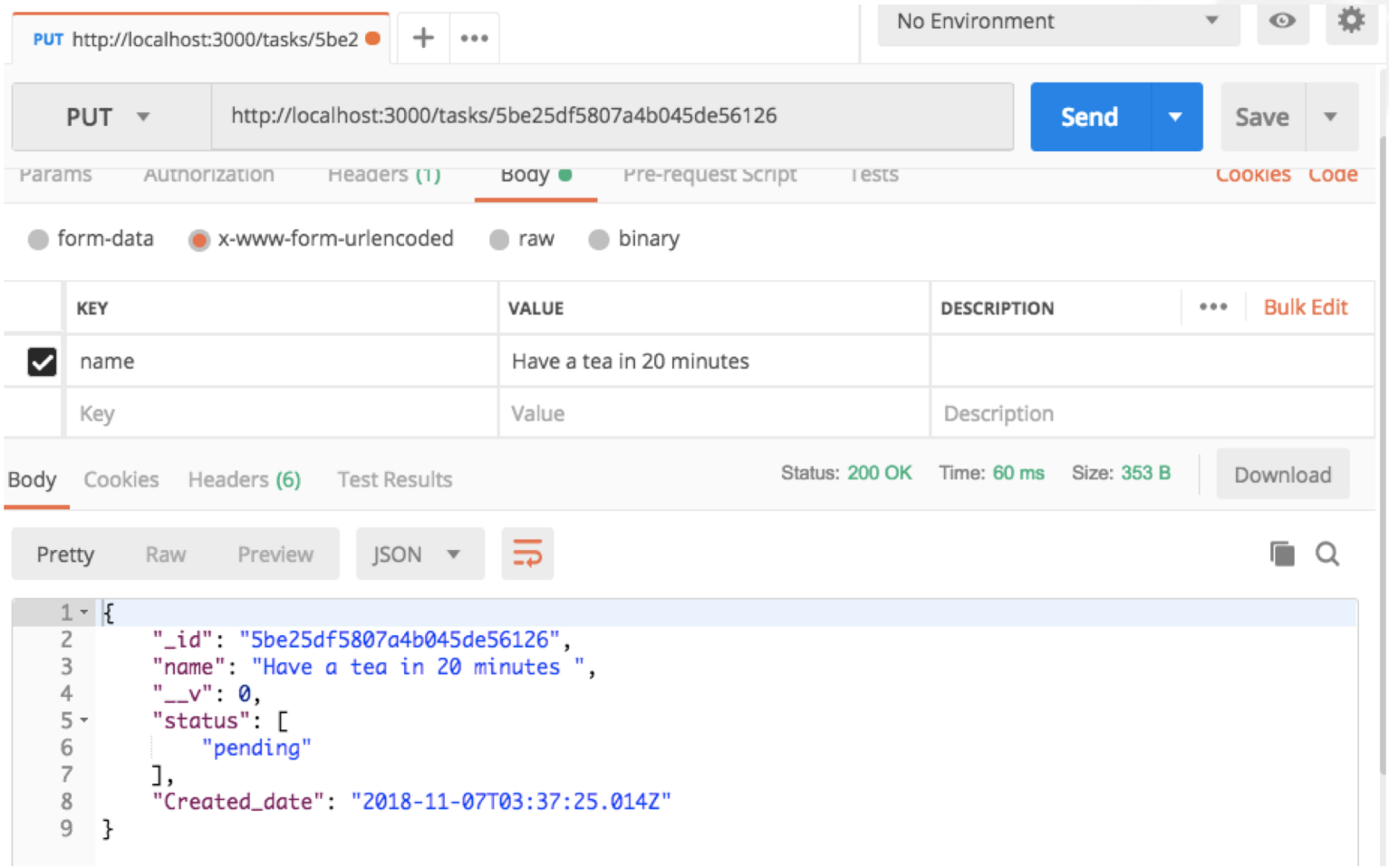
http://localhost:3000/tasks

using GET to list all tasks

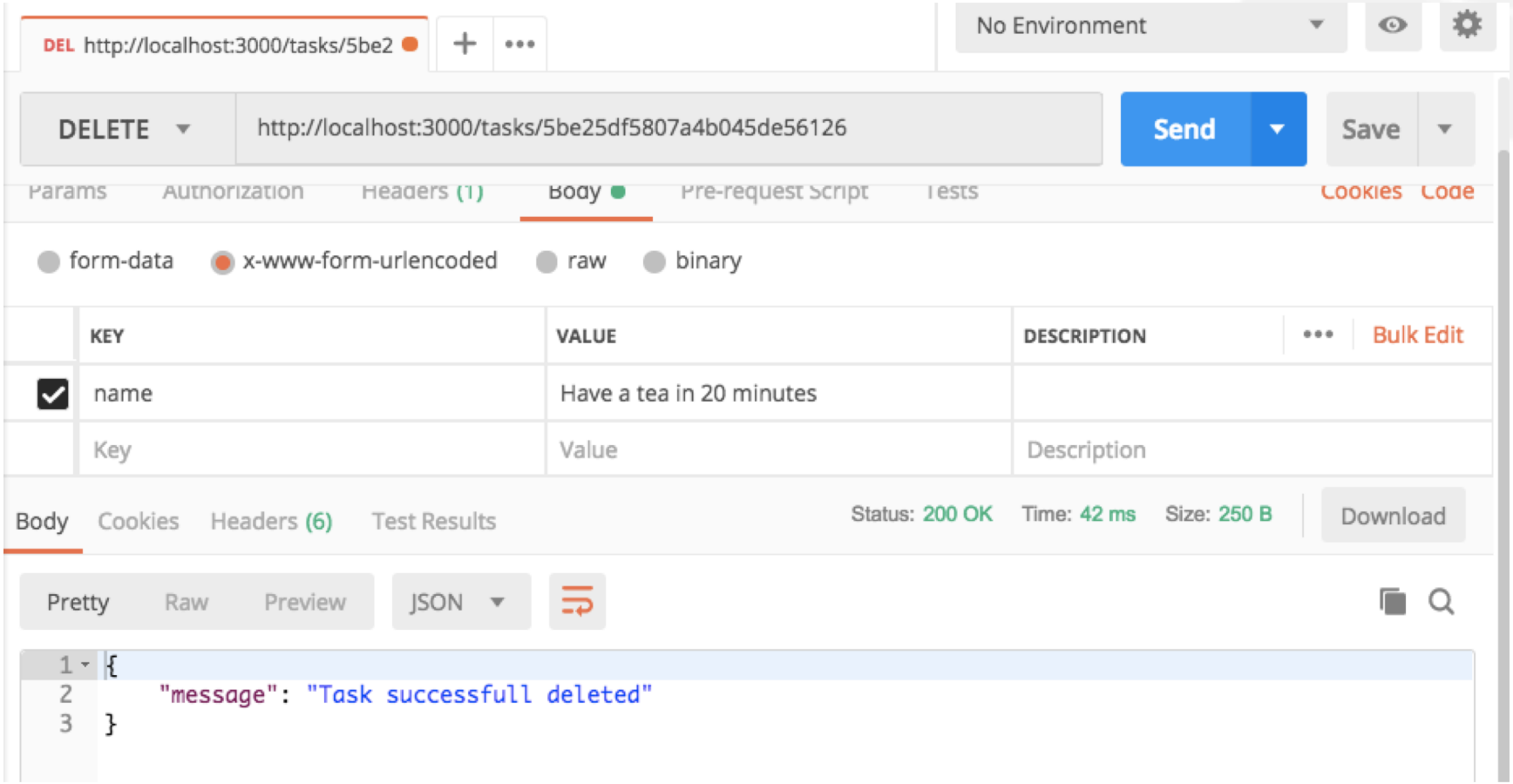


http://localhost:3000/tasks/<id>

using PUT to update a task

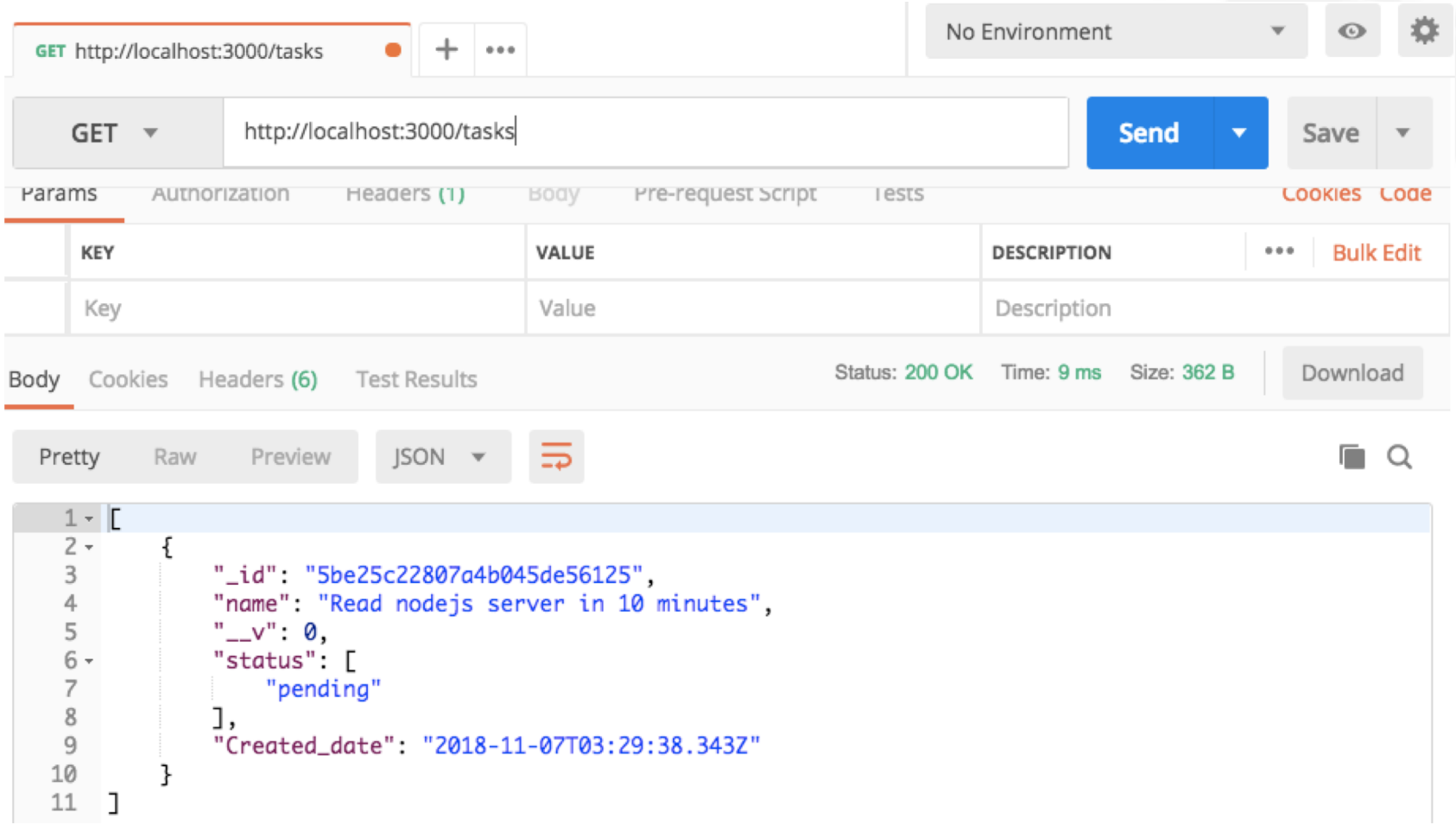


http://localhost:3000/tasks/<id>
using DELETE to delete a task



http://localhost:3000/tasks

using GET to list all tasks



References

- https://www.codementor.io/olatundegaruba/nodejs-restful-apis-in-10-minutes-q0sgsfhbd
- https://www.npmjs.com/package/mongoose
- https://www.restapitutorial.com/lessons/whatisrest.html
- https://medium.com/@purposenigeria/build-a-restful-api-with-node-js-and-express-js-d7e59c7a3dfb