Olatunde Garuba FOLLOW
Software Developer

# Build Node.js RESTful APIs in 10 Minutes

Published Jan 12, 2017

## 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 in his doctoral dissertation and defines the basis of RESTful-style as:

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

270

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; base URL, URL, media types, etc.

In this tutorial, we will learn how to create a RESTful API using Node.js.

### Tools:

- Node.js
- MongoDB
- Text editor (Atom, Sublime, etc) *(Read more: Best Text Editor? Atom vs Sublime vs Visual Studio Code vs Vim)*
- 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 (i.e. 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

I 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.

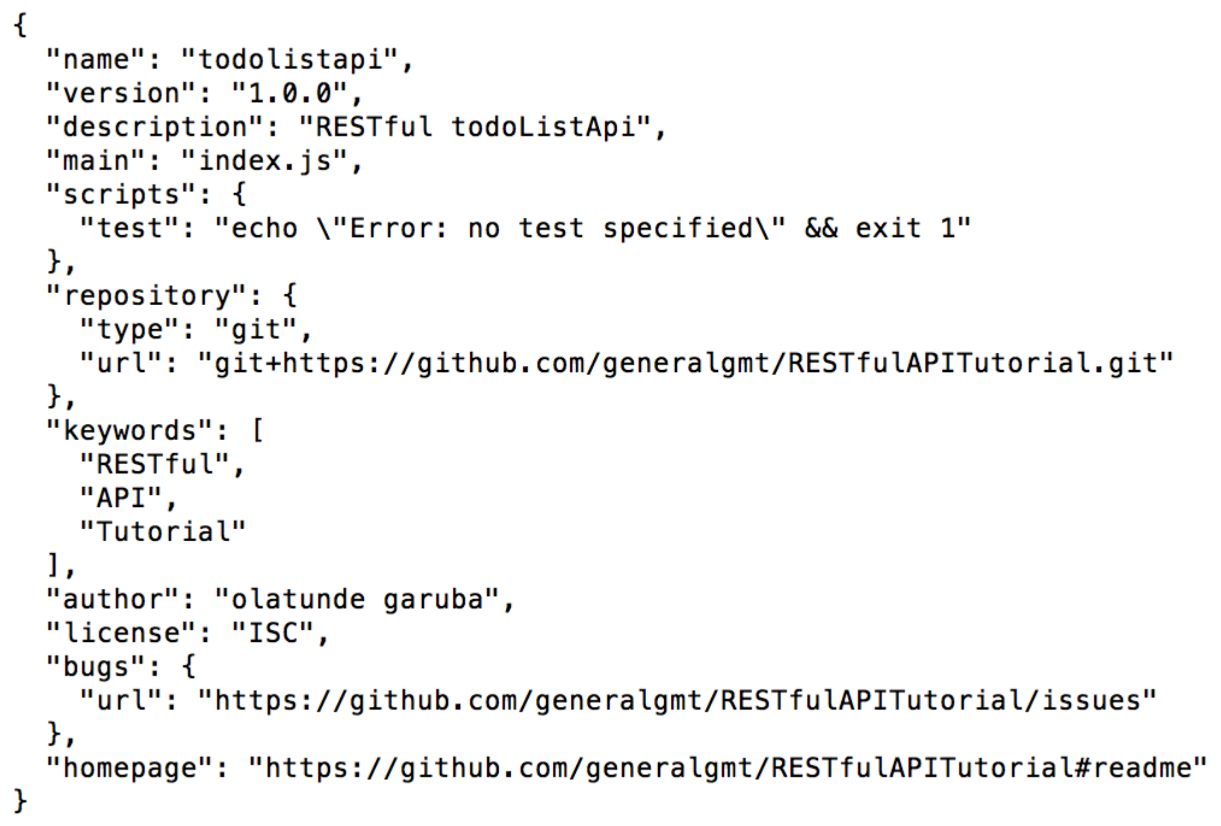
If you don't have it installed, kindly go through this link on how to install it in

♡ 270

If you do have Node and MongoDB installed, let's begin the tutorial with the following basic steps.

Open your terminal and kindly follow the following steps

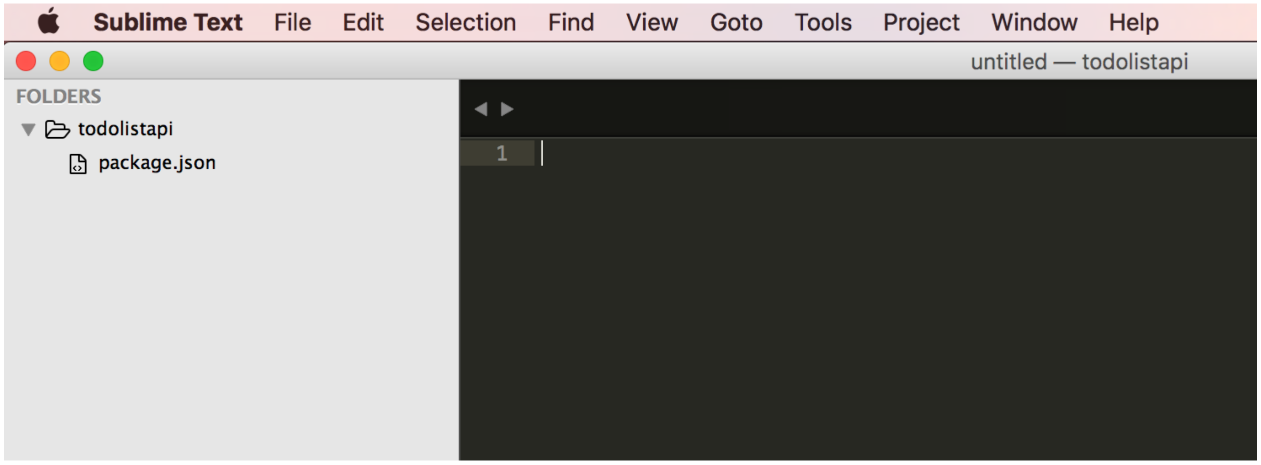
1. Create a Folder name todoListApi - `mkdir todoListApi`
2. Navigate to the root of your newly created folder - `cd todoListApi`
3. Create a package.json file - `npm init`
   Package.json is a file that gives the necessary information to npm which allows it to identify the project as well as handle the project's dependencies.
   npm init will prompt you to enter some information such as the app name, description, version, author, keyword and also ask if what you see is what you like.
   You should have something like this eventually.



```
{
  "name": "todolistapi",
  "version": "1.0.0",
  "description": "RESTful todoListApi",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/generalgmt/RESTfulAPITutorial.git"
  },
  "keywords": [
    "RESTful",
    "API",
    "Tutorial"
  ],
  "author": "olatunde garuba",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/generalgmt/RESTfulAPITutorial/issues"
  },
  "homepage": "https://github.com/generalgmt/RESTfulAPITutorial#readme"
}
```

270

Kindly type yes and press enter to complete the creation of our package.json. Having done all these, your folder structure should look like this:

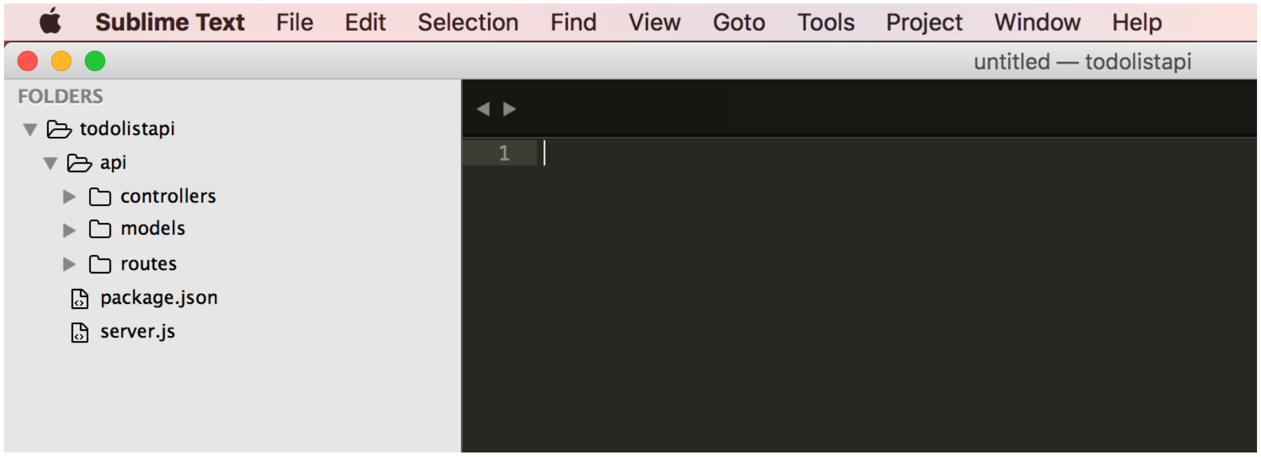

Sublime Text File Edit Selection Find View Goto Tools Project Window Help
untitled — todolistapi
FOLDERS
todolistapi
package.json

4. Create a file called server.js - `touch server.js` .

   In this server, we will writing the protocols to create our server.

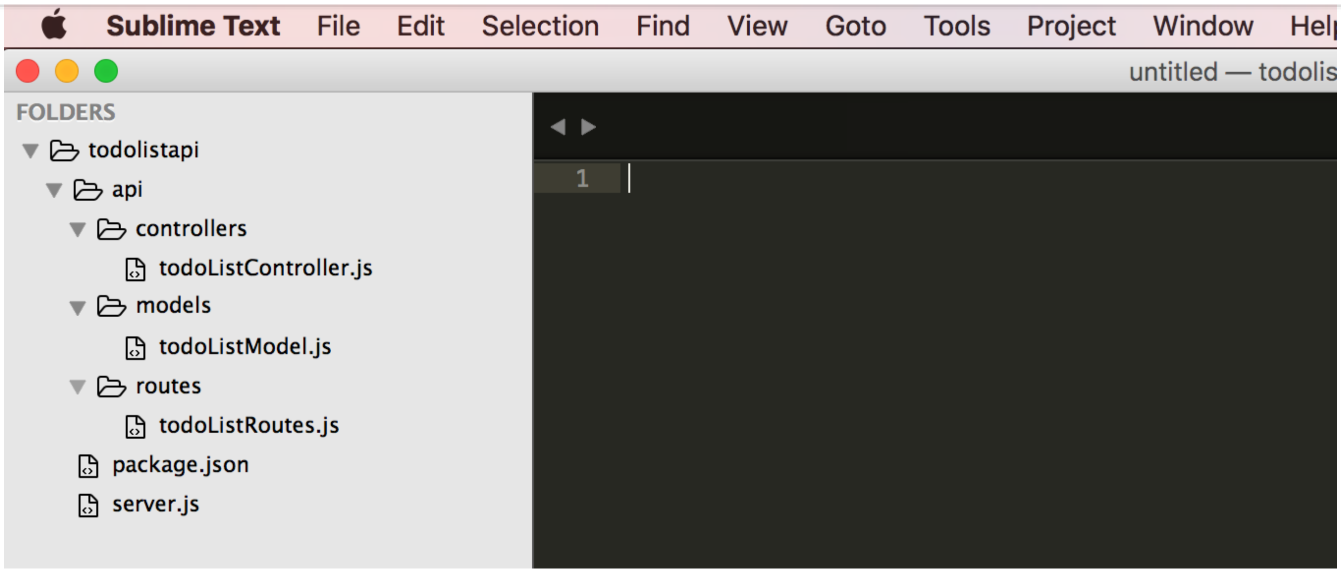
5. Create a folder called api - `mkdir api`

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



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/models/todoListModel.js api/routes/todoListRoutes.js`

270



## Server setup

Let's install express and nodmon, express will be used to create the server while nodmon 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
```

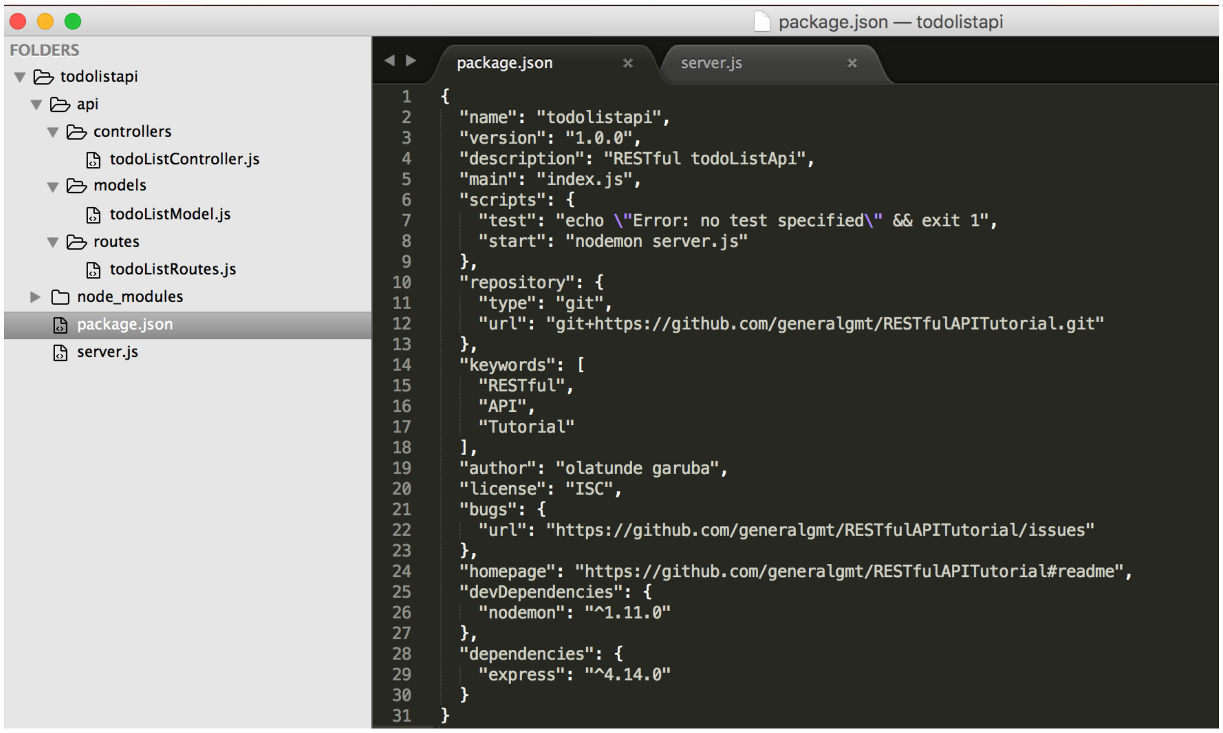
```
npm install express --save
```

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

1. Open the package.json file and add this task to the script

♡ 270

*"start": "nodemon server.js"*



```
{
  "name": "todolistapi",
  "version": "1.0.0",
  "description": "RESTful todoListApi",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon server.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/generalgmt/RESTfulAPITutorial.git"
  },
  "keywords": [
    "RESTful",
    "API",
    "Tutorial"
  ],
  "author": "olatunde garuba",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/generalgmt/RESTfulAPITutorial/issues"
  },
  "homepage": "https://github.com/generalgmt/RESTfulAPITutorial#readme",
  "devDependencies": {
    "nodemon": "^1.11.0"
  },
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

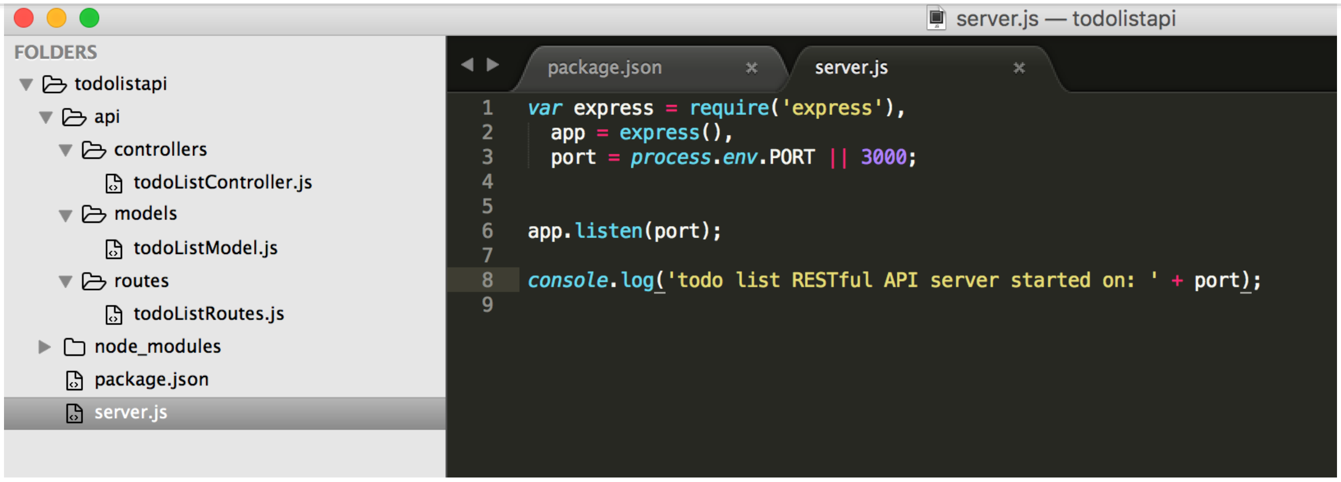
2. Open the server.js file and type/copy the code below into it

```
var express = require('express'),
  app = express(),
  port = process.env.PORT || 3000;

app.listen(port);

console.log('todo list RESTful API server started on: ' + port);
```

270



```
var express = require('express'),
  app = express(),
  port = process.env.PORT || 3000;


app.listen(port);

console.log('todo list RESTful API server started on: ' + port);
```

3.On your terminal, run `npm run start` this will start the server and then you will see

> *todo list RESTful API server started on: 3000*

## Setting up the schema

Firsr of all, let's install mongoose - `npm install mongoose --save`

### Why Mongoose?

Mongoose 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.

270

```
'use strict';
var mongoose = require('mongoose');
var Schema = mongoose.Schema;


var TaskSchema = new Schema({
  name: {
    type: String,
    required: 'Kindly enter the name of the task'
  },
  Created_date: {
    type: Date,
    default: Date.now
  },
  status: {
    type: [{
      type: String,
      enum: ['pending', 'ongoing', 'completed']
    }],
    default: ['pending']
  }
});

module.exports = mongoose.model('Tasks', TaskSchema);
```

From the code above, we required the mongoose in our file and then, we create a model of how our collection should look like.
As you can see, it the task collection(table) will contain a name: a string, and the date it was created. It also contains task status which we have defined as pending - a default value for every task created.

## Setting up the routes

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

270

Each of our routes has different route handler functions, which are executed when the route is matched.
Below we have defined two basic routes('/tasks', and '/tasks/taskId') with different methods
'/tasks' has to methods('GET' and 'POST'), while '/tasks/taskId' has GET, PUT and DELETE.
As you can see, we required the controller so each of the routes methods can call it's respective handler function.

To do this, open the todoListRoutes.js file in the route folder and paste the code snippet below into

```
'use strict';
module.exports = function(app) {
  var todoList = require('../controllers/todoListController');

  // todoList Routes
  app.route('/tasks')
    .get(todoList.list_all_tasks)
    .post(todoList.create_a_task);


  app.route('/tasks/:taskId')
    .get(todoList.read_a_task)
    .put(todoList.update_a_task)
    .delete(todoList.delete_a_task);
};
```

## Setting up the controller

Open todoListController.js file with your text editor( Sublime, Atom e.t.c) and let's deep dive into coding.

270

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. We will exported each of the functions for us to use in our routes.
Each of these functions uses different mongoose methods such as find, findById, findOneAndUpdate, save and remove.

```
'use strict';


var mongoose = require('mongoose'),
  Task = mongoose.model('Tasks');

exports.list_all_tasks = function(req, res) {
  Task.find({}, function(err, task) {
    if (err)
      res.send(err);
    res.json(task);
  });
};




exports.create_a_task = function(req, res) {
  var new_task = new Task(req.body);
  new_task.save(function(err, task) {
    if (err)
      res.send(err);
    res.json(task);
  });
};


exports.read_a_task = function(req, res) {
  Task.findById(req.params.taskId, function(err, task) {
    if (err)
      res.send(err);
    res.json(task);
  });
```

270

```
exports.update_a_task = function(req, res) {
  Task.findOneAndUpdate({_id: req.params.taskId}, req.body, {new: true}, functi
    if (err)
      res.send(err);
    res.json(task);
  });
};


exports.delete_a_task = function(req, res) {


  Task.remove({
    _id: req.params.taskId
  }, function(err, task) {
    if (err)
      res.send(err);
    res.json({ message: 'Task successfully deleted' });
  });
};
```

## Putting everything together

Earlier on, we had a minimal code for our server to be up and running in the server.js file.
In this section we will be connecting our handlers(controllers), database, the created models, body parser and the created routes together.

Open the server.js file created awhile ago and follow the following steps to put everything together.
Essentially, you will be replacing the code in your server.js with the code snippet from this section

270

1. Connect your database by adding a url to the mongoose instance connection
2. Load the created model - task
3. Install bodyParser and use
   bodyParser Parse incoming request bodies in a middleware before your handlers, available under the req.body property.
   It exposes various factories to create middlewares. All middlewares will populate the req.bodyproperty with the parsed body, or an empty object ({}) if there was no body to parse (or an error was returned).
4. Register our created routes in the server

270

```
var express = require('express'),
  app = express(),
  port = process.env.PORT || 3000,
  mongoose = require('mongoose'),
  Task = require('./api/models/todoListModel'), //created model loading here
  bodyParser = require('body-parser');

// mongoose instance connection url connection
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost/Tododb');


app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());


var routes = require('./api/routes/todoListRoutes'); //importing route
routes(app); //register the route


app.listen(port);


console.log('todo list RESTful API server started on: ' + port);
```

5.Start MongoDB Server

**Enjoy this post?**

Open your terminal and run `mongod`

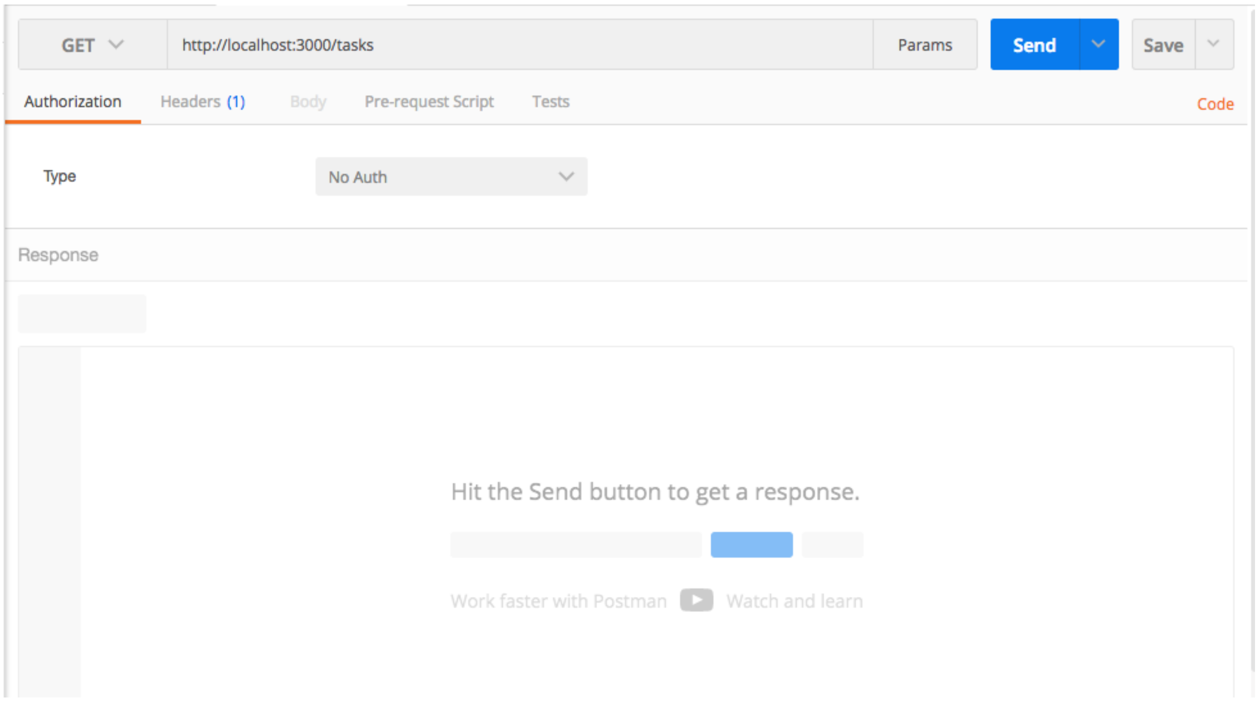
This will start your MongoDB server and then, node server could connect to the MongoDB instance. Once your MongoDB server is running, restart your node server by running: `rs` on your nodemon running terminal.

## Testing via Postman

270

Open your postman and type:

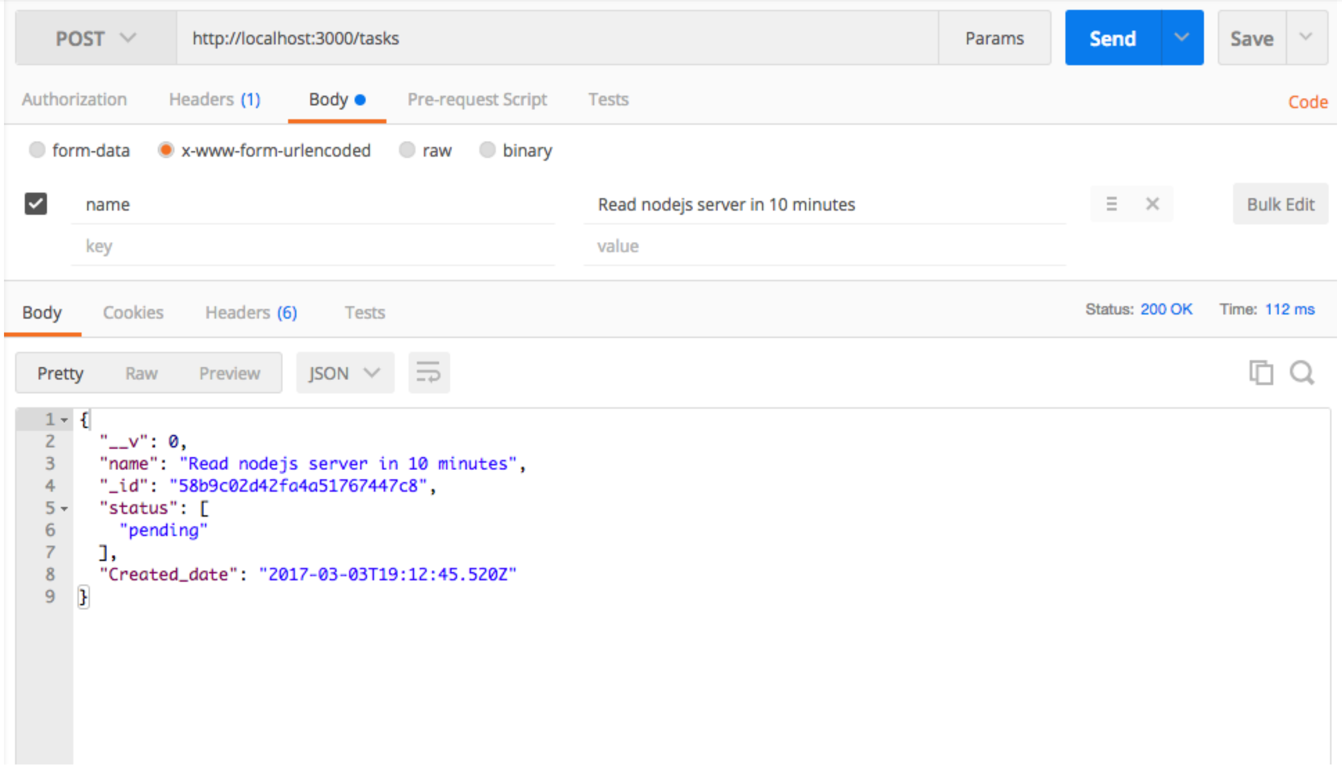
1. http://localhost:3000/tasks in the enter request URL section and press enter.

   On enter, you should see “[]” because there is nothing in the database yet.

2. 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

♡ 270



```
POST http://localhost:3000/tasks
Body: x-www-form-urlencoded
name    Read nodejs server in 10 minutes
key     value

Status: 200 OK    Time: 112 ms

{
  "__v": 0,
  "name": "Read nodejs server in 10 minutes",
  "_id": "58b9c02d42fa4a51767447c8",
  "status": [
    "pending"
  ],
  "Created_date": "2017-03-03T19:12:45.520Z"
}
```

## Adding a middleware

Having done all these, what happens if we entered a wrong route? say you entered 'http://localhost:3000/task', It responds with a message “Cannot GET /task”. Let’s add express middleware which could be used to return more interactive messages.

Middlewares basically intercepts incoming http request and as such you can use them to perform several operations ranging from authentication to validations etc.

To do this, open your server.js file and paste the code snippet into it.

```
app.use(function(req, res) {
  res.status(404).send({url: req.originalUrl + ' not found'})
});
```

270

The snippet above helps to redirect and respond whenever a wrong route is entered on the site.

Express Node.js Mongoose Restful web services JavaScript

Enjoy this post? Give **Olatunde Garuba** a like if it's helpful.

270 192 SHARE

**Olatunde Garuba**

Software Developer

Time has made me appreciate the fact that limitations are mere products of crude imaginations and thinking, which forms my belief that “all thing are possible". Possibilities and solutions abound around us. Just think it, and rea...

FOLLOW

## 192 Replies

Leave a reply

Z **Ziegler** 14 days ago

270

Thanks a lot for ur work

Reply

**Petar Petrović** 15 days ago

Hi everyone,

I was wondering about wrapping Controller in a class. Does it make sense?

Thank you (:

Reply

**Zaid Wahab** 13 days ago

may be this video help you :)
https://www.youtube.com/watch?v=WDrU305J1yw

1 Reply

**Petar Petrović** 13 days ago

Thanks.
I am going now through all the videos of Maximilian’s guide.

Reply

**Antonin Jubault** 20 days ago

Awesome tutorial man.

Reply

Show more replies

270

LEARN MORE

Andrei Neagoie

## Don’t be a Junior Developer □□

Seriously, don’t be a junior developer. A junior developer puts this title in their resume, emails, and LinkedIn… They announce it to the world. Don’t.

When you do that, this is what recruiters and companies see: *“Hi, I’m desperately looking to get hired as a developer. I’m still new at this, but can you please please please place a bet on me and hope that I turn out to be an asset and not a liability for your company. Oh, and I’m also going to need a lot of help from your staff for the first six months!”*

READ MORE

270

♡ 270