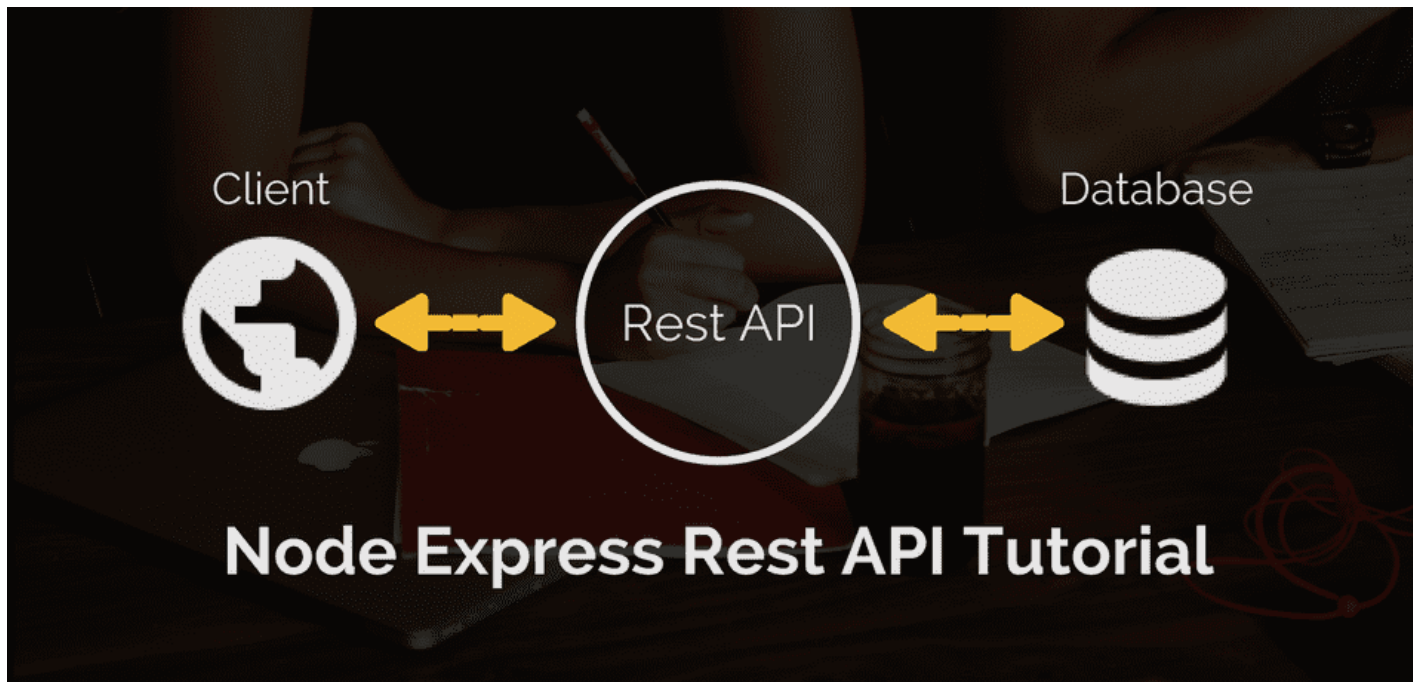



Building a Restful CRUD API with Node.js, Express and MongoDB

by RAJEEV SINGH · NODE JS · 4 MINS





In this tutorial, we'll be building a RESTful CRUD (Create, Retrieve, Update, Delete) API with Node.js, Express and MongoDB. We'll use Mongoose for interacting with the MongoDB instance.

Express is one of the most popular web frameworks for node.js. It is built on top of node.js http module, and adds support for routing, middleware, view system etc. It is very simple and minimal, unlike other frameworks that try to do way too much, thereby reducing the flexibility for developers to have their own design choices.

Mongoose is an ODM (Object Document Mapping) tool for Node.js and MongoDB. It helps you convert the objects in your code to documents in the database and vice versa.

Before proceeding to the next section, Please install MongoDB in your machine if you have not done already. Checkout the [official MongoDB installation manual](#) for any help with the installation.

Our Application

In this tutorial, We will be building a simple Note-Taking application. We will build Rest APIs for creating, listing, editing and deleting a Note.

We'll start by building a simple web server and then move on to configuring the database, building the `Note` model and different routes for handling all the CRUD operations.

Finally, we'll test our REST APIs using Postman.

Also, In this post, we'll heavily use ES6 features like `let`, `const`, `arrow functions`, `promises` etc. It's good to familiarize yourself with these features. I

recommend [this re-introduction to Javascript](#) to brush up these concepts.

Well! Now that we know what we are going to build, We need a cool name for our application. Let's call our application `EasyNotes`.

Creating the Application

1. Fire up your terminal and create a new folder for the application.

```
$ mkdir node-easy-notes-app
```

2. Initialize the application with a package.json file

Go to the root folder of your application and type `npm init` to initialize your app with a `package.json` file.

```
$ cd node-easy-notes-app  
$ npm init
```

```
name: (node-easy-notes-app)
version: (1.0.0)
description: Never miss a thing in Life. Take notes quickly. Organ
entry point: (index.js) server.js
test command:
git repository:
keywords: Express RestAPI MongoDB Mongoose Notes
author: callicoder
license: (ISC) MIT
About to write to /Users/rajeevkumarsingh/node-easy-notes-app/pack
```

```
{
  "name": "node-easy-notes-app",
  "version": "1.0.0",
  "description": "Never miss a thing in Life. Take notes quickly.",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Express",
```

```
    "RestAPI",
    "MongoDB",
    "Mongoose",
    "Notes"
  ],
  "author": "callicoder",
  "license": "MIT"
}
```

Is this ok? (yes) **yes**

Note that I've specified a file named `server.js` as the entry point of our application. We'll create `server.js` file in the next section.

3. Install dependencies

We will need `express`, `mongoose` and `body-parser` modules in our application. Let's install them by typing the following command -

```
$ npm install express body-parser mongoose --save
```

I've used `--save` option to save all the dependencies in the `package.json` file.
The final `package.json` file looks like this -

```
{
  "name": "node-easy-notes-app",
  "version": "1.0.0",
  "description": "Never miss a thing in Life. Take notes quickly.",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Express",
    "RestAPI",
    "MongoDB",
    "Mongoose",
    "Notes"
  ],
  "author": "callicoder",
  "license": "MIT",
  "dependencies": {
    "body-parser": "^1.18.3",
```

```
"express": "^4.16.3",  
"mongoose": "^5.2.8"  
}  
}
```

Our application folder now has a `package.json` file and a `node_modules` folder -

```
node-easy-notes-app  
├── node_modules/  
└── package.json
```

Setting up the web server

Let's now create the main entry point of our application. Create a new file named `server.js` in the root folder of the application with the following contents -

```
const express = require('express');  
const bodyParser = require('body-parser');  
  
// create express app
```



```
const app = express();

// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

// parse requests of content-type - application/json
app.use(bodyParser.json());

// define a simple route
app.get('/', (req, res) => {
    res.json({ "message": "Welcome to EasyNotes application. Take n
});

// listen for requests
app.listen(3000, () => {
    console.log("Server is listening on port 3000");
});
```

First, We import express and body-parser modules. [Express](#), as you know, is a web framework that we'll be using for building the REST APIs, and [body-parser](#) is a

module that parses the request (of various content types) and creates a `req.body` object that we can access in our routes.

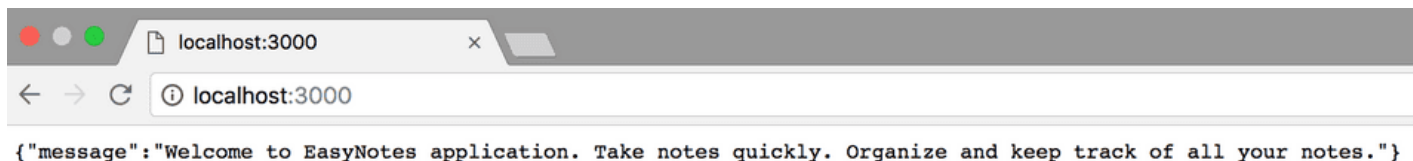
Then, We create an express app, and add two `body-parser` middlewares using express's `app.use()` method. A `middleware` is a function that has access to the `request` and `response` objects. It can execute any code, transform the request object, or return a response.

Then, We define a simple `GET` route which returns a welcome message to the clients.

Finally, We listen on port 3000 for incoming connections.

All right! Let's now run the server and go to `http://localhost:3000` to access the route we just defined.

```
$ node server.js  
Server is listening on port 3000
```



Configuring and Connecting to the database

I like to keep all the configurations for the app in a separate folder. Let's create a new folder `config` in the root folder of our application for keeping all the configurations -

```
$ mkdir config  
$ cd config
```

Now, Create a new file `database.config.js` inside `config` folder with the following contents -

```
module.exports = {  
  url: 'mongodb://localhost:27017/easy-notes'  
}
```

We'll now import the above database configuration in `server.js` and connect to the database using mongoose.

Add the following code to the `server.js` file after `app.use(bodyParser.json())` line -

```
// Configuring the database
const dbConfig = require('./config/database.config.js');
const mongoose = require('mongoose');

mongoose.Promise = global.Promise;

// Connecting to the database
mongoose.connect(dbConfig.url, {
  useNewUrlParser: true
}).then(() => {
  console.log("Successfully connected to the database");
}).catch(err => {
  console.log('Could not connect to the database. Exiting now...');
  process.exit();
});
```

Please run the server and make sure that you're able to connect to the database -

```
$ node server.js  
Server is listening on port 3000  
Successfully connected to the database
```

Defining the Note model in Mongoose

Next, We will define the `Note` model. Create a new folder called `app` inside the root folder of the application, then create another folder called `models` inside the `app` folder -

```
$ mkdir -p app/models  
$ cd app/models
```

Now, create a file called `note.model.js` inside `app/models` folder with the following contents -

```
const mongoose = require('mongoose');  
  
const NoteSchema = mongoose.Schema({  
  title: String,  
  content: String  
}, {  
  timestamps: true  
});  
  
module.exports = mongoose.model('Note', NoteSchema);
```

The `Note` model is very simple. It contains a `title` and a `content` field. I have also added a `timestamps` option to the schema.

Mongoose uses this option to automatically add two new fields - `createdAt` and `updatedAt` to the schema.

Defining Routes using Express

Next up is the routes for the Notes APIs. Create a new folder called `routes` inside the `app` folder.

```
$ mkdir app/routes  
$ cd app/routes
```

Now, create a new file called `note.routes.js` inside `app/routes` folder with the following contents -

```
module.exports = (app) => {  
  const notes = require('../controllers/note.controller.js');  
  
  // Create a new Note  
  app.post('/notes', notes.create);  
  
  // Retrieve all Notes
```

```
app.get('/notes', notes.findAll);

// Retrieve a single Note with noteId
app.get('/notes/:noteId', notes.findOne);

// Update a Note with noteId
app.put('/notes/:noteId', notes.update);

// Delete a Note with noteId
app.delete('/notes/:noteId', notes.delete);
}
```

Note that We have added a `require` statement for `note.controller.js` file. We'll define the controller file in the next section. The controller will contain methods for handling all the CRUD operations.

Before defining the controller, let's first include the routes in `server.js`. Add the following `require` statement before `app.listen()` line inside `server.js` file.

```
// .....
```



```
// Require Notes routes
require('./app/routes/note.routes.js')(app);

// .....
```

If you run the server now, you'll get the following error -

```
$ node server.js
module.js:472
    throw err;
    ^

Error: Cannot find module '../controllers/note.controller.js'
```

This is because we haven't defined the controller yet. Let's do that now.

Writing the Controller functions

Create a new folder called `controllers` inside the `app` folder, then create a new file called `note.controller.js` inside `app/controllers` folder with the following contents -

```
const Note = require('../models/note.model.js');

// Create and Save a new Note
exports.create = (req, res) => {

};

// Retrieve and return all notes from the database.
exports.findAll = (req, res) => {

};

// Find a single note with a noteId
exports.findOne = (req, res) => {

};

// Update a note identified by the noteId in the request
exports.update = (req, res) => {

};
```

```
// Delete a note with the specified noteId in the request
exports.delete = (req, res) => {

};
```

Let's now look at the implementation of the above controller functions one by one

-

Creating a new Note

```
// Create and Save a new Note
exports.create = (req, res) => {
  // Validate request
  if(!req.body.content) {
    return res.status(400).send({
      message: "Note content can not be empty"
    });
  }

  // Create a Note
  const note = new Note({
```

```
        title: req.body.title || "Untitled Note",
        content: req.body.content
    });

    // Save Note in the database
    note.save()
        .then(data => {
            res.send(data);
        }).catch(err => {
            res.status(500).send({
                message: err.message || "Some error occurred while cre
            });
        });
    };
};
```

Retrieving all Notes

```
// Retrieve and return all notes from the database.
exports.findAll = (req, res) => {
    Note.find()
        .then(notes => {
```

```
        res.send(notes);
    }).catch(err => {
        res.status(500).send({
            message: err.message || "Some error occurred while ret
        });
    });
};
```

Retrieving a single Note

```
// Find a single note with a noteId
exports.findOne = (req, res) => {
    Note.findById(req.params.noteId)
        .then(note => {
            if(!note) {
                return res.status(404).send({
                    message: "Note not found with id " + req.params.no
                });
            }
            res.send(note);
        }).catch(err => {
```

```
    if(err.kind === 'ObjectId') {
      return res.status(404).send({
        message: "Note not found with id " + req.params.no
      });
    }
    return res.status(500).send({
      message: "Error retrieving note with id " + req.params
    });
  });
};
```

Updating a Note

```
// Update a note identified by the noteId in the request
exports.update = (req, res) => {
  // Validate Request
  if(!req.body.content) {
    return res.status(400).send({
      message: "Note content can not be empty"
    });
  }
}
```

```
// Find note and update it with the request body
Note.findByIdAndUpdate(req.params.noteId, {
  title: req.body.title || "Untitled Note",
  content: req.body.content
}, {new: true})
.then(note => {
  if(!note) {
    return res.status(404).send({
      message: "Note not found with id " + req.params.no
    });
  }
  res.send(note);
}).catch(err => {
  if(err.kind === 'ObjectId') {
    return res.status(404).send({
      message: "Note not found with id " + req.params.no
    });
  }
  return res.status(500).send({
    message: "Error updating note with id " + req.params.n
  });
});
```

```
});  
};
```

The `{new: true}` option in the `findByIdAndUpdate()` method is used to return the modified document to the `then()` function instead of the original.

Deleting a Note

```
// Delete a note with the specified noteId in the request  
exports.delete = (req, res) => {  
  Note.findByIdAndRemove(req.params.noteId)  
    .then(note => {  
      if(!note) {  
        return res.status(404).send({  
          message: "Note not found with id " + req.params.no  
        });  
      }  
      res.send({message: "Note deleted successfully!"});  
    }).catch(err => {  
      if(err.kind === 'ObjectId' || err.name === 'NotFound') {  
        return res.status(404).send({
```



```
        message: "Note not found with id " + req.params.id
    });
}
return res.status(500).send({
    message: "Could not delete note with id " + req.params.id
});
});
};
```

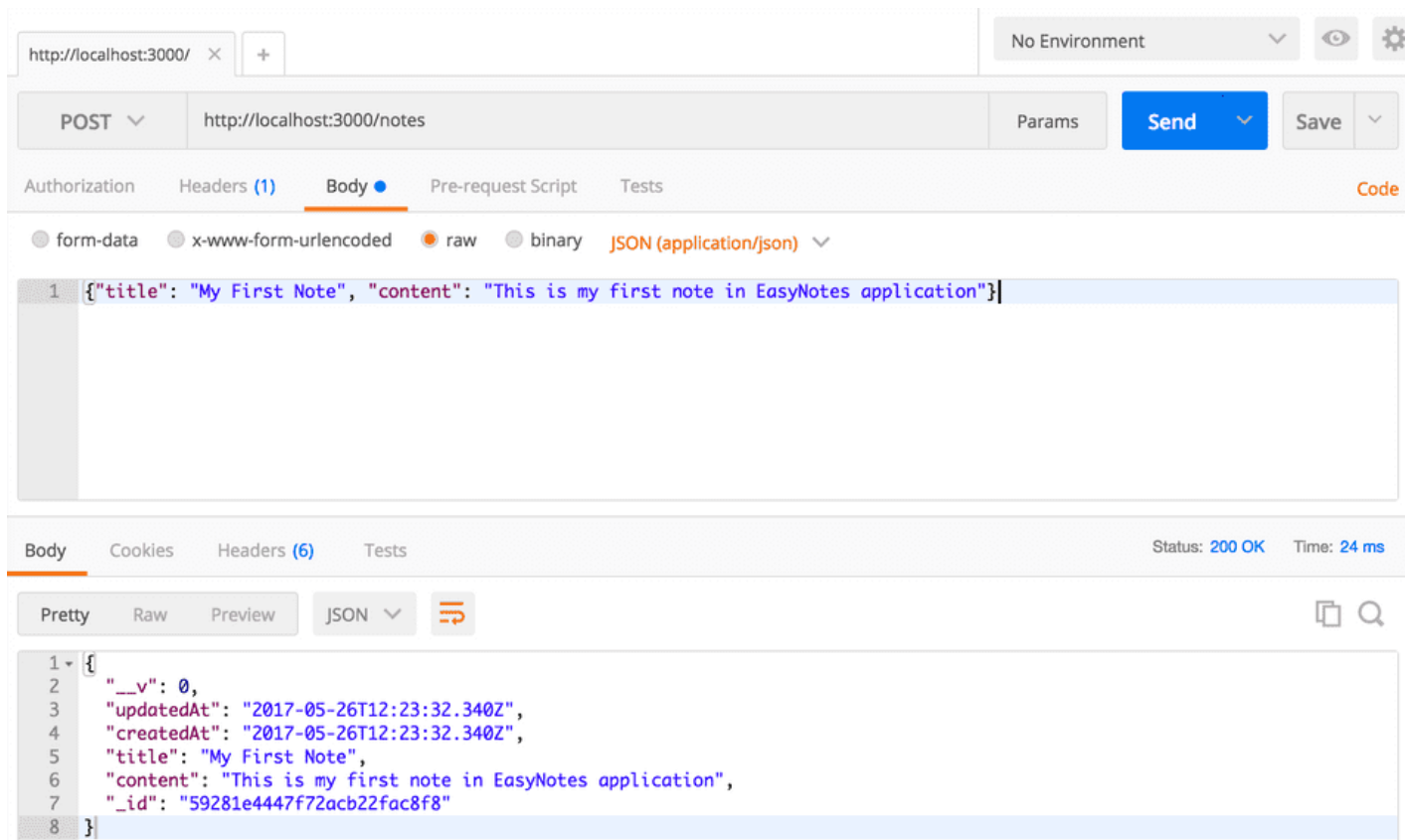
You can check out the documentation of all the methods that we used in the above APIs on Mongoose's official documentation -

- [Mongoose save\(\)](#)
- [Mongoose find\(\)](#)
- [Mongoose findById\(\)](#)
- [Mongoose findByIdAndUpdate\(\)](#)
- [Mongoose findByIdAndRemove\(\)](#)

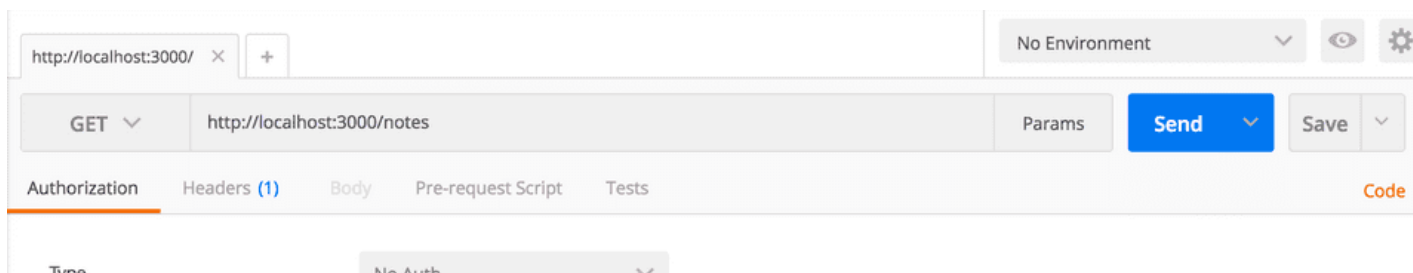
Testing our APIs

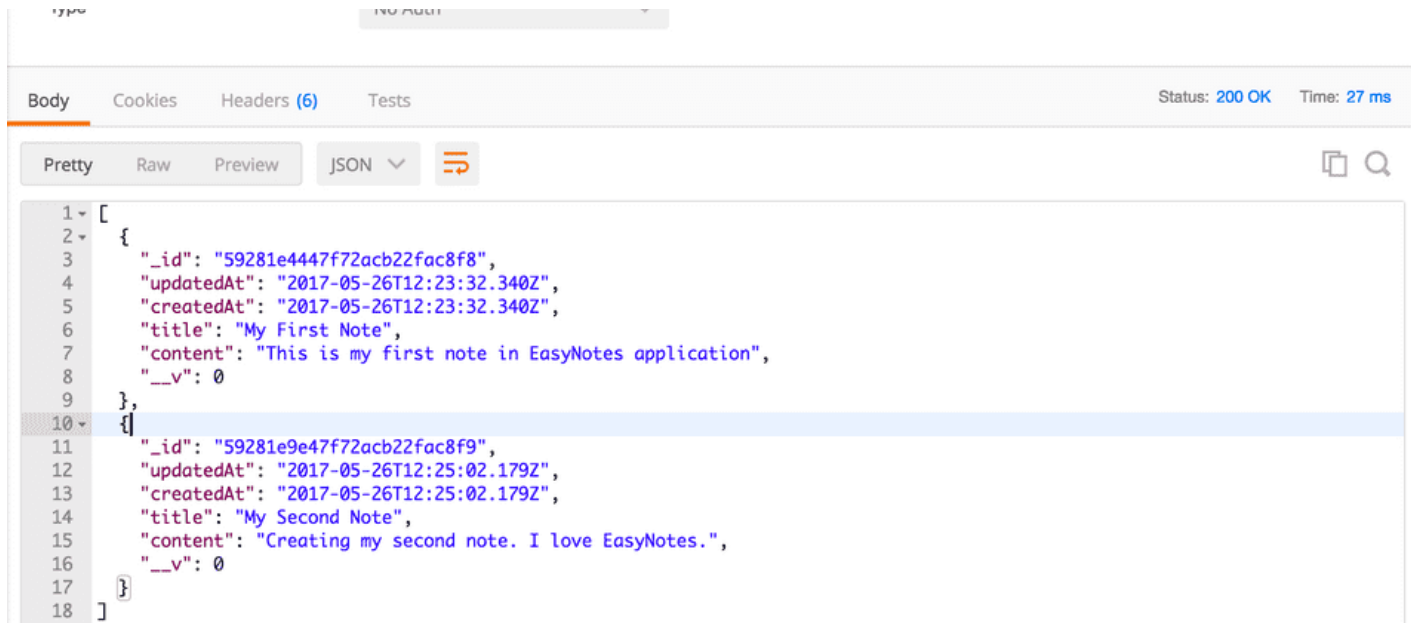
Let's now test all the APIs one by one using postman.

Creating a new Note using POST /notes API



Retrieving all Notes using GET /notes API



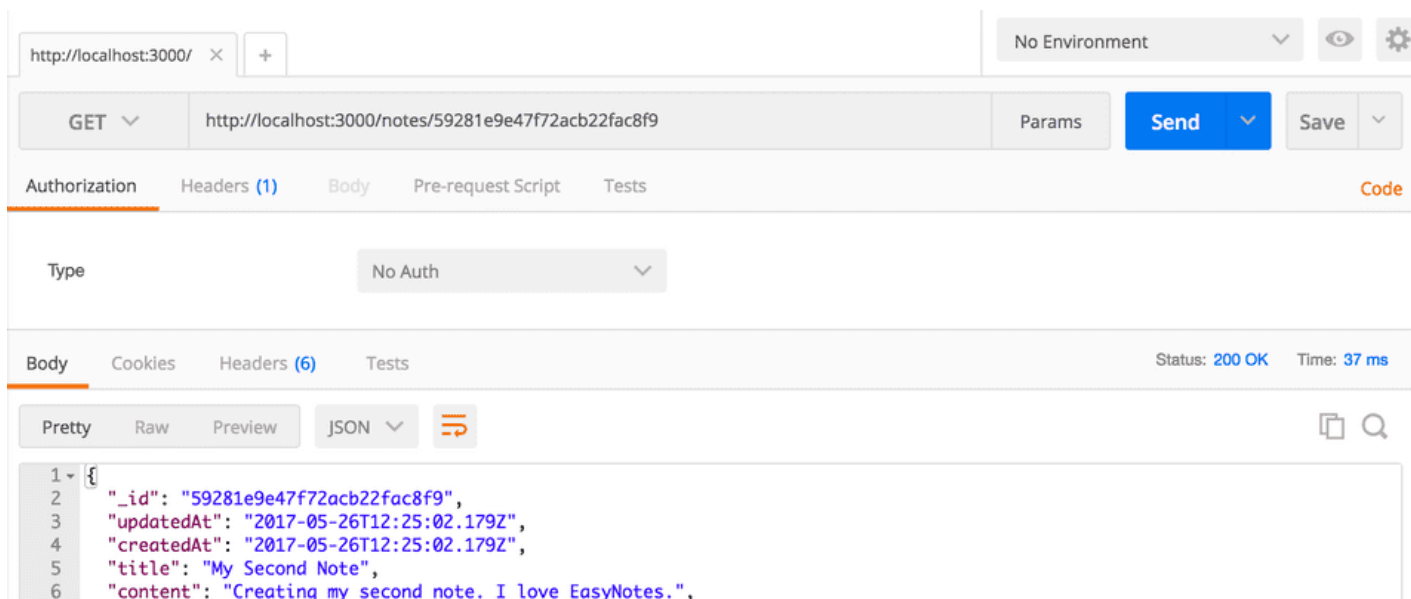


The screenshot shows a REST client interface with the following details:

- Body tab:** Selected, showing a JSON array of two note objects.
- Status:** 200 OK
- Time:** 27 ms
- JSON View:** Pretty-printed.
- JSON Data:**

```
1 [
2   {
3     "_id": "59281e4447f72acb22fac8f8",
4     "updatedAt": "2017-05-26T12:23:32.340Z",
5     "createdAt": "2017-05-26T12:23:32.340Z",
6     "title": "My First Note",
7     "content": "This is my first note in EasyNotes application",
8     "__v": 0
9   },
10  {
11    "_id": "59281e9e47f72acb22fac8f9",
12    "updatedAt": "2017-05-26T12:25:02.179Z",
13    "createdAt": "2017-05-26T12:25:02.179Z",
14    "title": "My Second Note",
15    "content": "Creating my second note. I love EasyNotes.",
16    "__v": 0
17  }
18 ]
```

Retrieving a single Note using `GET /notes/:noteId` API



The screenshot shows a REST client interface with the following details:

- Request:**
 - Method:** GET
 - URL:** `http://localhost:3000/notes/59281e9e47f72acb22fac8f9`
 - Authorization:** No Auth
- Response:**
 - Status:** 200 OK
 - Time:** 37 ms
 - JSON View:** Pretty-printed.
 - JSON Data:**

```
1 {
2   "_id": "59281e9e47f72acb22fac8f9",
3   "updatedAt": "2017-05-26T12:25:02.179Z",
4   "createdAt": "2017-05-26T12:25:02.179Z",
5   "title": "My Second Note",
6   "content": "Creating my second note. I love EasyNotes.",
7 }
```

```
7   "__v": 0
8 }]
```

Updating a Note using PUT /notes/:noteId API

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:3000/notes/59281e9e47f72acb22fac8f9`
- Method:** PUT
- Body:**

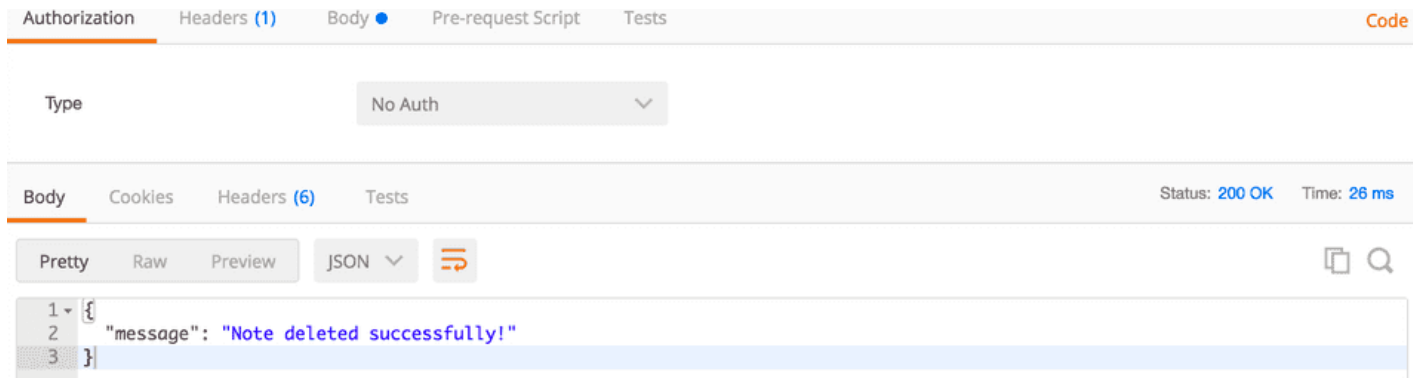
```
{ "title": "Edited Title of Second Note", "content": "Edited Content of Second Note." }
```
- Response:** Status 200 OK, Time 61 ms. The response body is shown in JSON format:

```
{  "_id": "59281e9e47f72acb22fac8f9",  "updatedAt": "2017-05-26T12:26:38.486Z",  "createdAt": "2017-05-26T12:25:02.179Z",  "title": "Edited Title of Second Note",  "content": "Edited Content of Second Note.",  "__v": 0 }
```

Deleting a Note using DELETE /notes/:noteId API

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:3000/notes/59281e9e47f72acb22fac8f9`
- Method:** DELETE



Conclusion

In this tutorial, We learned how to build rest apis in node.js using express framework and mongodb.

You can find the code for this tutorial in [my github repository](#). Please ask any questions that you might have in the comment section below.

Thanks for reading. See you in the next tutorial!

Share on social media

 Facebook

 Twitter

 LinkedIn

 Reddit

ALSO ON CALLICODER

Spring Boot OAuth2 Social Login with ...
3 years ago • 91 comments
In this article, You'll learn how to add social as well as email and password ...

Spring Boot Actuator: Health check, ...
3 years ago • 7 comments
Spring Boot actuator helps you monitor and manage your Spring boot ...

Spring Boot File Upload / Download ...
3 years ago • 19 comments
In this article, you'll learn how to upload and download files in a ...

Spring Boot Social Login ...
3 years ago • ...
Build a full s with Spring containing s

185 Comments CalliCoder Disqus' Privacy Policy

1 Login ▾

Recommend 32

Tweet

Share

Sort by Newest ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS



Name



Tiffany Chong • 4 months ago

Awesome tutorial :)

^ | ▾ • Reply • Share ›



Lusavya Saidi • 5 months ago

hey i need help

^ | ▾ • Reply • Share ›



\$@k!ndu • 7 months ago

Thanks, so Helpful!!!

1 ^ | v • Reply • Share ›



ALRVD • 9 months ago

Why you use 2 error handlers, one with 500 err code and one with 404 err code, when the noteld note found?

^ | v 1 • Reply • Share ›



Pasha Kolesnyk • a year ago

Really easy.. Ulike java with spring boot

^ | v • Reply • Share ›



ALRVD → Pasha Kolesnyk • 9 months ago

Java is headache

^ | v • Reply • Share ›



Sameer Karmacharya • a year ago

Thank you very very much. It was very helpful.

1 ^ | v • Reply • Share ›



Curryandketchup Norway • a year ago

Thank u so much! Such a good explanation, useful for beginneers..

1 ^ | v • Reply • Share ›



nasir kiyani • a year ago • edited

Thank u so much sir!

if any one faced empty array issue when he call list method.Than please also add third parameter "collection name" in the last line of your model class.

```
module.exports = mongoose.model('Note', NoteSchema, "your collection name");
```



```
module.exports = mongoose.model( note , noteSchema, your collection name );
```

Thank u.

2 ^ | v • Reply • Share ›



deepak verma • a year ago

nice tutorial sir

1 ^ | v • Reply • Share ›



Aman Gupta • a year ago

Please also provide us signup and login also

2 ^ | v 1 • Reply • Share ›



Aman Gupta • a year ago

Can we see mongo Database where my query execute..

^ | v • Reply • Share ›



Rajnikant Chaudhary • a year ago

Very usefull

^ | v • Reply • Share ›



Mohamed Afzal Mulla • a year ago

Great post! Thanks Callicode! I'm subscribe to your newsletter. PS. Anyone trying this, you must have mongodb working on localhost. Callicoder, will be great if you had a part 2 that showed connect this to a form on Create-react-app

^ | v • Reply • Share ›



Mufiq Patoni • 2 years ago

wow its nice tutorial..

TQ

^ | v • Reply • Share ›



ajay konda • 2 years ago

Sir please tell me how to debug , I am new to nodejs and very well aware of .net api and postman , please reply as quick as can

^ | v • Reply • Share ›



Сафонов из Назарета → ajay konda • 2 years ago

VS Code has built-in node debugger



^ | v • Reply • Share ›



Aruljothy sundaramoorthy • 2 years ago

Awsome . Just nailed it



Awesome..just nailed it

1 ^ | v • Reply • Share ›



Abdur Rahim • 2 years ago

Where do they save when we create a new note.

1 ^ | v • Reply • Share ›



Ferdinand von Schirach • 2 years ago

One of the best tutorials I've seen so far about this topic. I would really like to see how to build a React front end to this API.

^ | v • Reply • Share ›



Anshuman • 2 years ago

Hi

a little tip was working on new server and wasted 15 mins as db wasn't connecting to the node app before i realised its not running
systemctl status mongod

^ | v • Reply • Share ›



aishwarya bhosale • 2 years ago

Hii I am getting this following error:

throw new Error(msg);

^

Error: Route.get() requires a callback function but got a [object Undefined]
at Route.(anonymous function) [as get] (/home/gslab/Desktop/user-app/node_modules/express/lib/router/route.js:202:15)
at [Function.app](#).(anonymous function) [as get] (/home/gslab/Desktop/user-app/node_modules/express/lib/application.js:482:19)
at module.exports (/home/gslab/Desktop/user-app/app/routes/user.routes.js:8:9)
at Object.<anonymous> (/home/gslab/Desktop/user-app/server.js:35:39)
at Module._compile (module.js:652:30)
at Object.Module._extensions.js (module.js:683:10)

```
at Object.Module._extensions..js (module.js:663:10)
at Module.load (module.js:565:32)
at tryModuleLoad (module.js:505:12)
at Function.Module._load (module.js:497:3)
at Function.Module.runMain (module.js:693:10)
at startup (bootstrap_node.js:188:16)
at bootstrap_node.js:609:3
```

Please help me out here

^ | v • Reply • Share ›



Hem Chavda → aishwarya bhosale • 2 years ago

Have you got solution above erro? if then share me please.

^ | v • Reply • Share ›



Dương Hải Quân • 2 years ago

my nodejs server allway connected mongodb with any url database in database.config.js . How it doesn't connected ?

^ | v • Reply • Share ›



Irfan Razzaq → Dương Hải Quân • 2 years ago

If you specify any url that is not in mongodb it will create an empty db for you with the name you have specified in url, but it will be empty.

^ | v • Reply • Share ›



Hubert Kanyamahanga • 2 years ago

Nice Tutorials!

^ | v • Reply • Share ›



Eric Bruno Nz • 2 years ago

Hello I am trying to test the API unfortunately I get 400 error. my DB is connected I am not sure what I am doing wrong.



^ | v • Reply • Share ›



ciao ciao → Eric Bruno Nz • 2 years ago • edited

You need to add headers Content-Type = application/json

2 ^ | v • Reply • Share ›



Anshuman → ciao ciao • 2 years ago

wow thanks !

^ | v • Reply • Share ›



Bruno Muniz • 2 years ago

Awesome!

^ | v • Reply • Share ›



Dmitriy Svirin • 2 years ago

This is really helpful! Than you so much!

^ | v • Reply • Share ›



Ashwin • 2 years ago

This is one of the best tutorials I have ever used. Fantastic. Thanks!!

^ | v • Reply • Share ›



DMongo • 2 years ago

The tutorial is great. Do you know how you could pass in a parameter to the api such as 'name' or 'serial number' to return a specific record instead of filtering by the MongoDB ':id' ?

3 ^ | v • Reply • Share ›



Sarvdip • 2 years ago

Thanks for tutorial. I have query that - How automatically created collection name notes?

^ | v • Reply • Share ›



Mubarak Show • 2 years ago • edited

I see lots of people struggling to run this project successfully.

Steps to run project

1. open new terminal and startup mongodb server: `mongod`
2. run the project: `npm start`

Please upvote if this helped so other may find it easily :)

3 ^ | v • Reply • Share ›



bikash shah • 2 years ago

thanks rajeev, loved your tutorial. Its really great to begin with :)

^ | v • Reply • Share ›



Kushagra Kiran • 2 years ago

TypeError: Note is not a constructor

at exports.create (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/app/controllers/note.controller.js:13:18)

at Layer.handle [as handle_request] (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/node_modules/express/lib/router/layer.js:95:5)

at next (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/node_modules/express/lib/router/route.js:137:13)

at Route.dispatch (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-

```
app/node_modules/express/lib/router/route.js:112:3)
at Layer.handle [as handle_request] (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/node_modules/express/lib/router/layer.js:95:5)
at /opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/node_modules/express/lib/router/index.js:281:22
at Function.process_params (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/node_modules/express/lib/router/index.js:335:12)
at next (/opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-app/node_modules/express/lib/router/index.js:275:10)
at /opt/lampp/htdocs/bitbucketworkspace/nodelearnings/node-easy-notes-
```

[see more](#)

^ | v • Reply • Share ›



Enes Varcana → Kushagra Kiran • 2 years ago

You should add this line to the end of note.model.js file.

```
module.exports = mongoose.model('Note', NoteSchema);
```

^ | v • Reply • Share ›



Pankaj Patel • 2 years ago

It's an amazing and easy example to understand, it's saved lots of time of mine.

^ | v • Reply • Share ›



Rajeev Singh Mod → Pankaj Patel • 2 years ago

Thank you :)

^ | v • Reply • Share ›



Abhishek Kumar • 2 years ago

How to secure these APIs ?

How can we restrict other users to access these apis ?

1 ^ | v 1 • Reply • Share ›



Jerry Sunny • 2 years ago

great stuff really helped a lot to start with

^ | v • Reply • Share ›



Salma Zaghloul • 2 years ago

thank you for the great tutorial i would like to know how can i document this api please using openApi. that would be really really helpful

^ | v • Reply • Share ›



lockwear • 2 years ago

Nice man! Great tutorial Helped me a lot

^ | v • Reply • Share ›



RamzanAli Momin • 2 years ago • edited

Thanks ! Very useful article for beginner

^ | v 1 • Reply • Share ›



gaurav • 3 years ago

Please help me how can i create signin and signup api?

^ | v • Reply • Share ›



Dev Aggrwal • 3 years ago • edited

api

^ | v • Reply • Share ›



Saravanan Chinnadurai • 3 years ago

Hi Master,

I am getting this error though i pass the values as you shown

Note content can not be empty

^ | v 1 • Reply • Share ›

^ | v • Reply • Share ›



Moulder_B → Saravanan Chinnadurai • 2 years ago

You need to ensure that Postman is running the Body and selecting the JSON(application/json) drop down item, not plain text--this is what fixed the error for me.

^ | v • Reply • Share ›



Salma Zaghloul → Saravanan Chinnadurai • 2 years ago

i keep getting the same problem

^ | v • Reply • Share ›

CalliCoder

Copyright © 2021 CalliCoder [Privacy Policy](#)

[Home](#)

[About](#)

[Contact](#)

[Sitemap](#)

[URL Encoder](#)

[URL Decoder](#)

[Base64 Encoder](#)

[Base64 Decoder](#)

[JSON Formatter](#)

[ASCII Table](#)