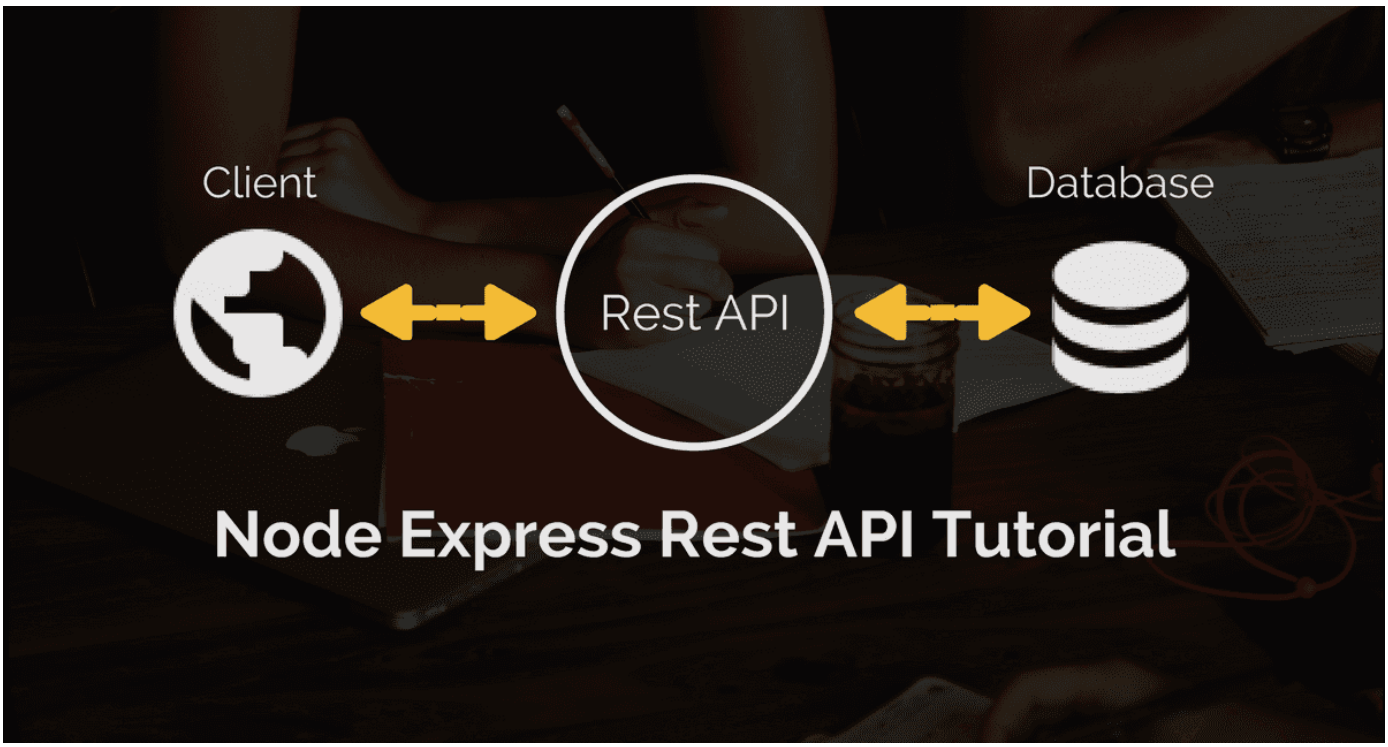
CALLICODER

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

by RAJEEV SINGH · NODE JS · JUNE 13, 2017 · 4 MINS READ



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 do way to 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 MogngoDB 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. Organize and
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/package.json

{
  "name": "node-easy-notes-app",
  "version": "1.0.0",
  "description": "Never miss a thing in Life. Take notes quickly. Organize
  "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. Organize
  "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 notes qui
});

// 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.

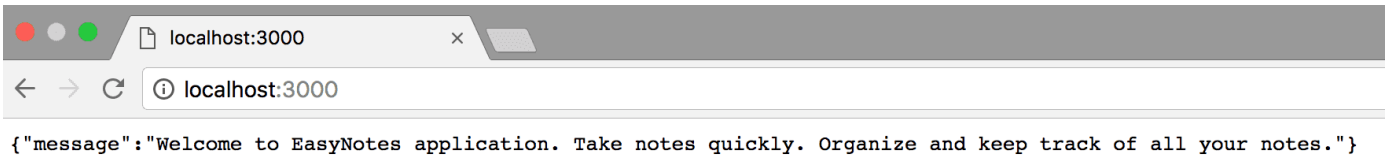
Then, We create an express app, and add two `body-parser` middlewares using express's `app.use()` method. A middleware is a function that has access to the `request` and `response` objects. It can execute any code, transform the request object, or return a response.

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...', err);
    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 creating th
        });
    });
};
```

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 retrieving
        });
    });
};
```

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.noteId
                });
            }
            res.send(note);
        }).catch(err => {
            if(err.kind === 'ObjectId') {
                return res.status(404).send({
                    message: "Note not found with id " + req.params.noteId
                });
            }
            return res.status(500).send({
                message: "Error retrieving note with id " + req.params.noteId
            });
        });
    };
```

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.noteId
```

```
            });
        }
        res.send(note);
    }).catch(err => {
        if(err.kind === 'ObjectId') {
            return res.status(404).send({
                message: "Note not found with id " + req.params.noteId
            });
        }
        return res.status(500).send({
            message: "Error updating note with id " + req.params.noteId
        });
    });
};
```

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

```
        });
    });
};
```

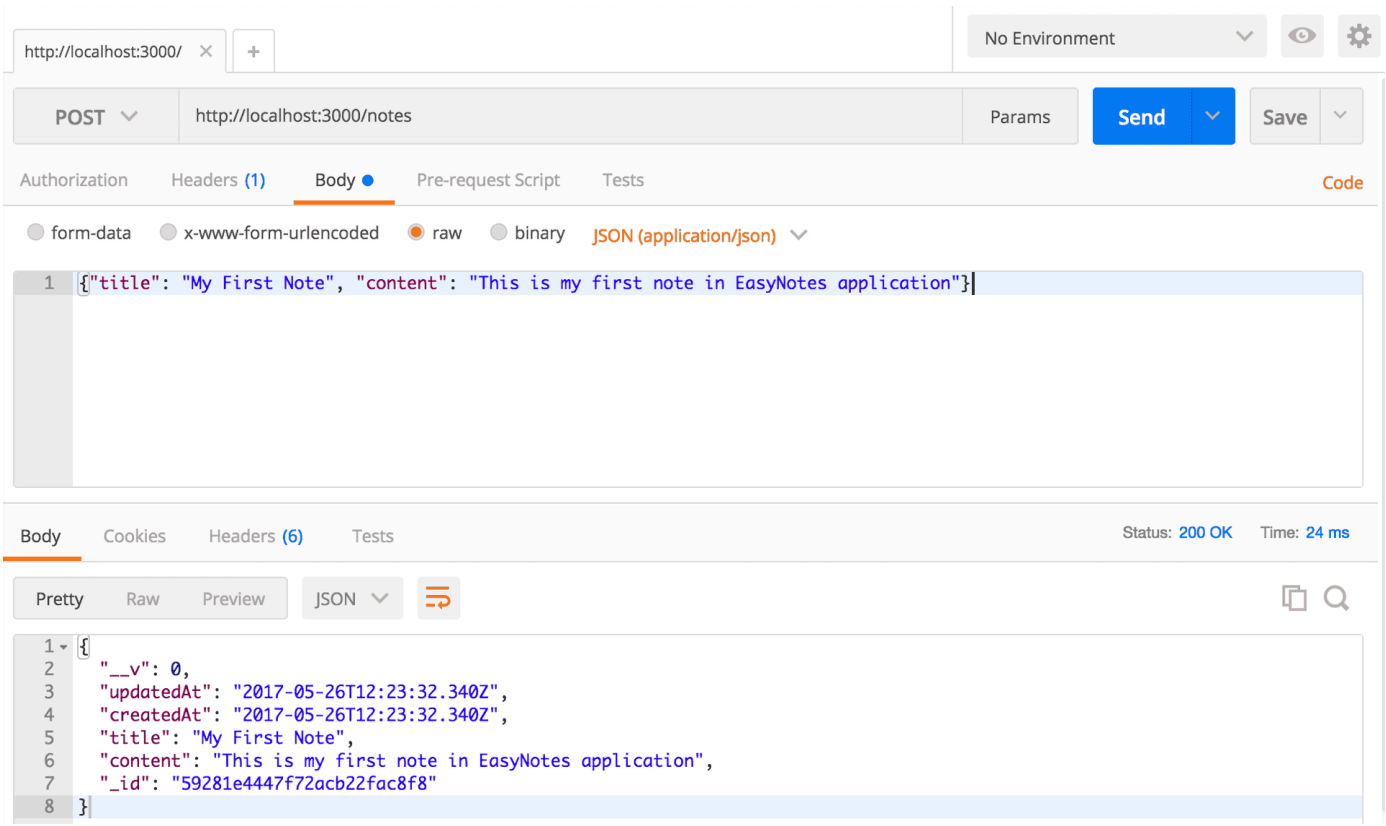
You can check out the documentation of all the methods that we used in the above APIs on Mongoose's official documentation -

- Mongoose save()
- Mongoose find()
- Mongoose findById()
- Mongoose findByIdAndUpdate()
- Mongoose findByIdAndRemove()

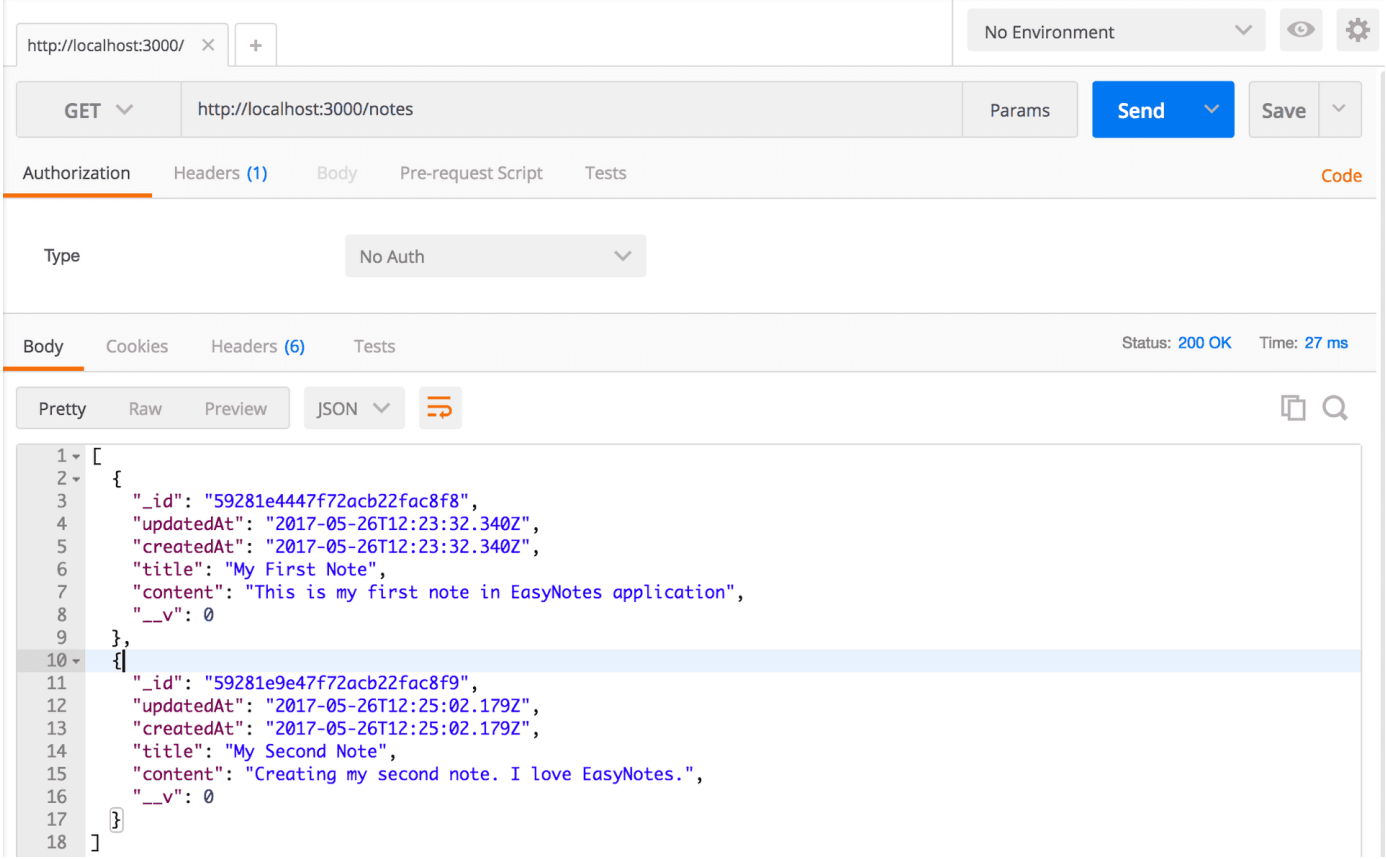
Testing our APIs

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

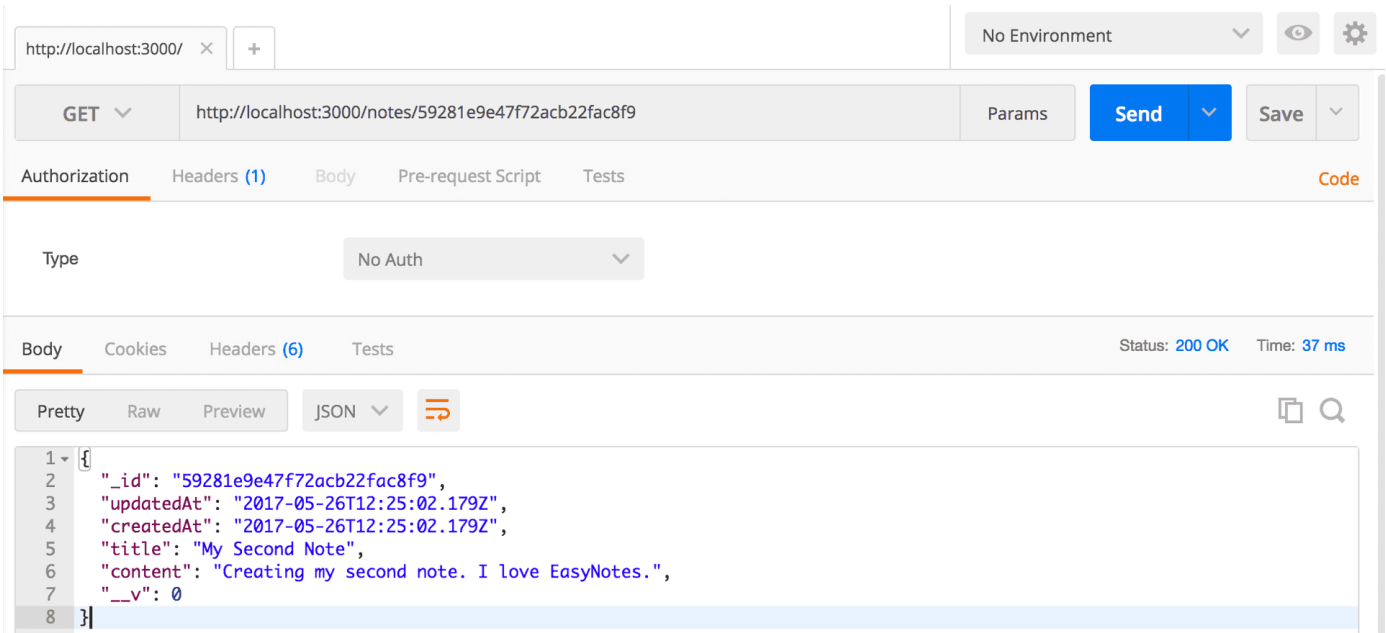
Creating a new Note using `POST /notes` API



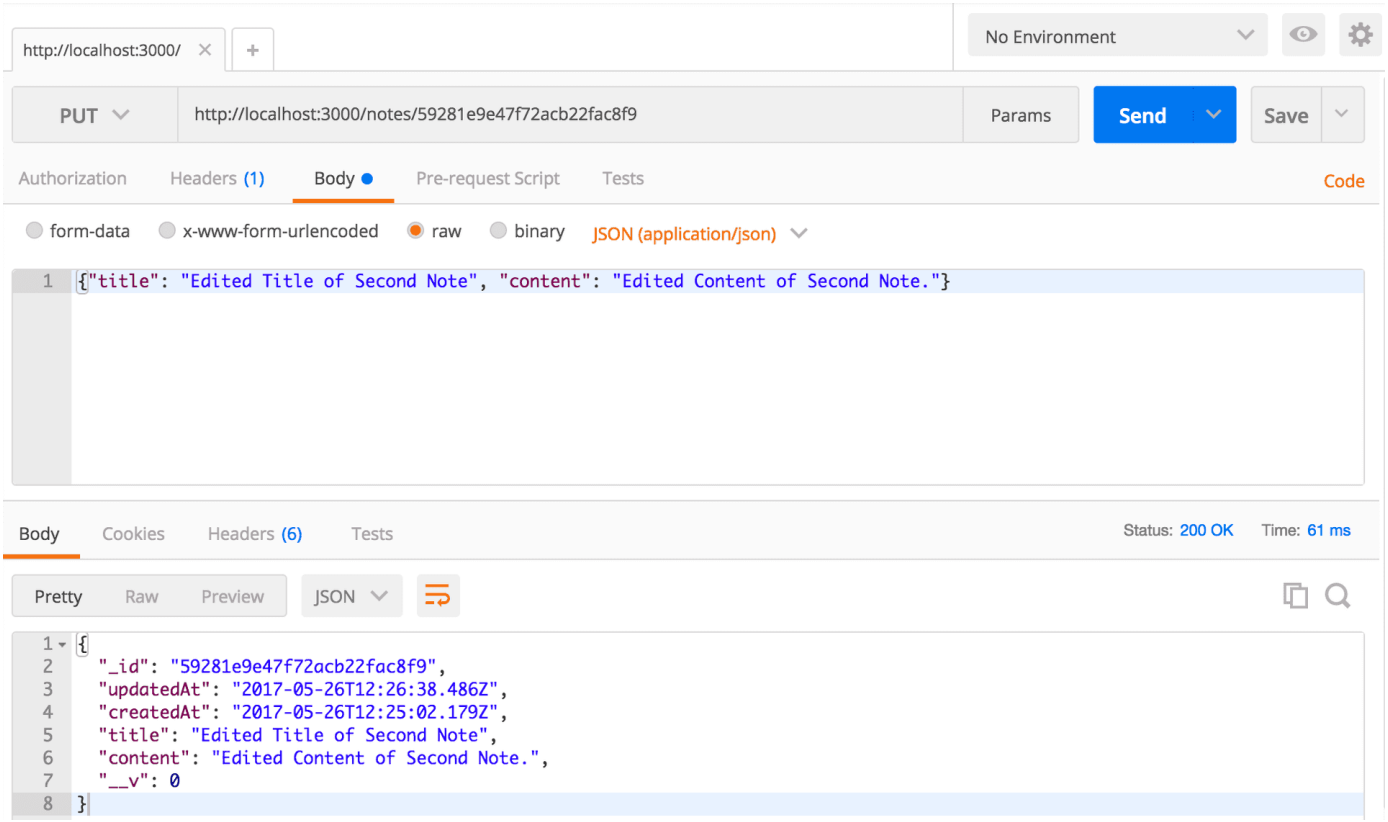
Retrieving all Notes using `GET /notes` API



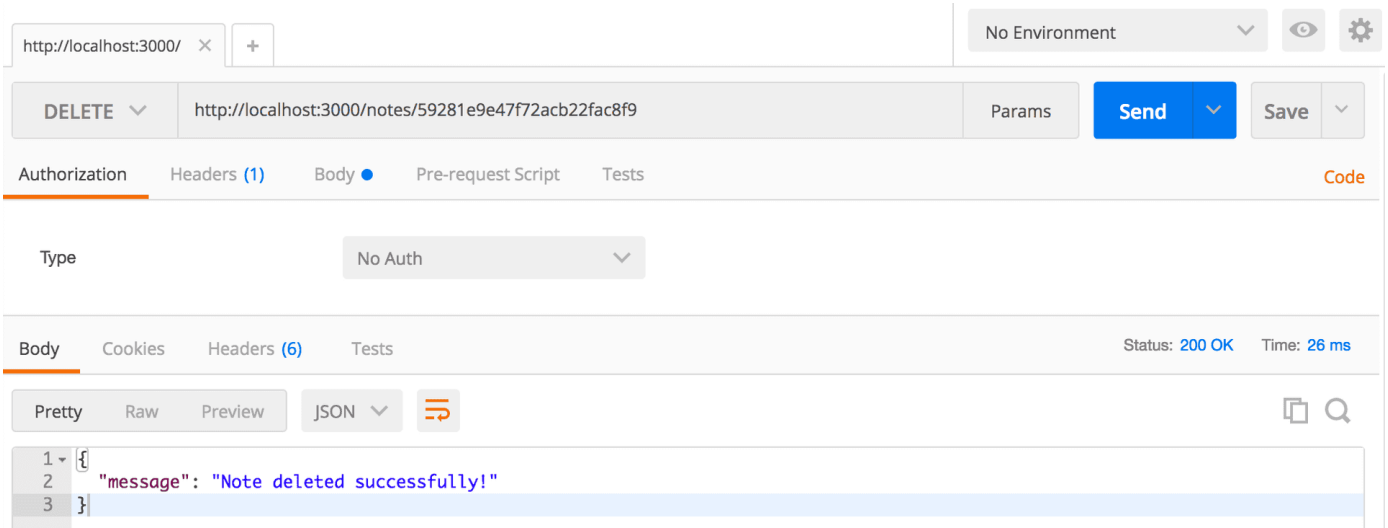
Retrieving a single Note using `GET /notes/:noteId` API



Updating a Note using `PUT /notes/:noteId` API



Deleting a Note using `DELETE /notes/:noteId` API



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!