We will be building a simple Note-Taking application. We will build Rest APIs for creating, listing, editing and deleting a Note. Basic RESTful CRUD (Create, Retrieve, Update, Delete) API with

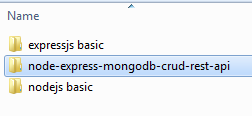
* Node.js,
* Express and
* MongoDB.

We’ll use Mongoose for interacting with the MongoDB instance.

[Mongoose](http://mongoosejs.com/) 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**.

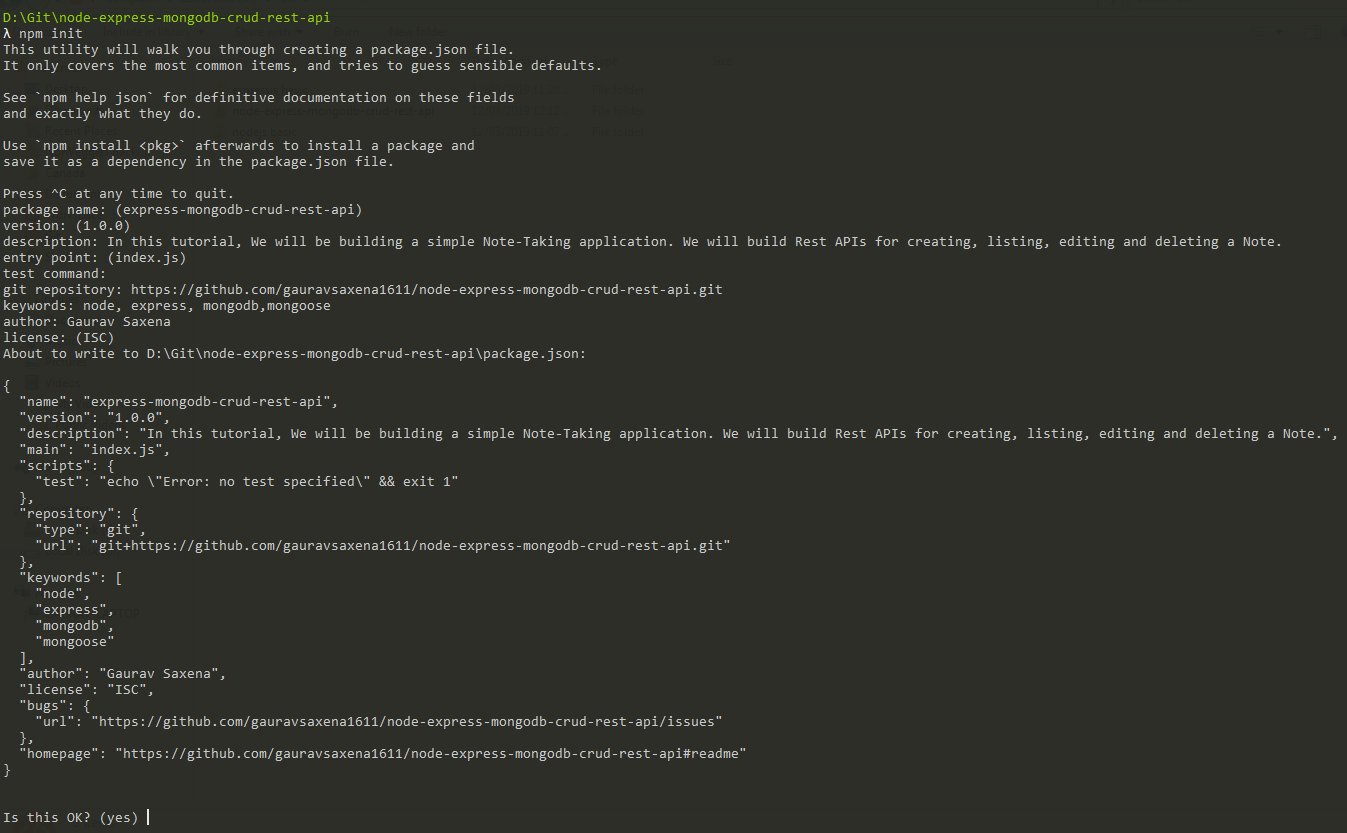
# Creating the Application

**1, create a new folder for the application**

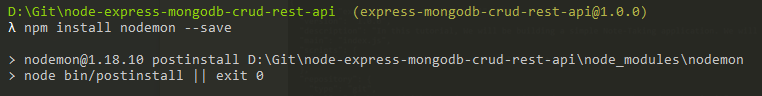


**2, Initialize the application with a package.json file.**

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

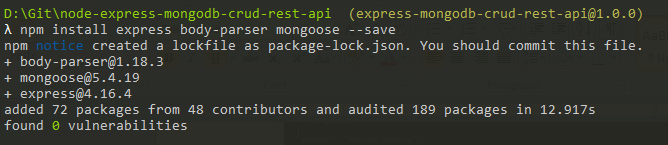


Also install nodemon module for devops



**3. Install dependencies**

We will need express, mongoose and body-parser modules in our application

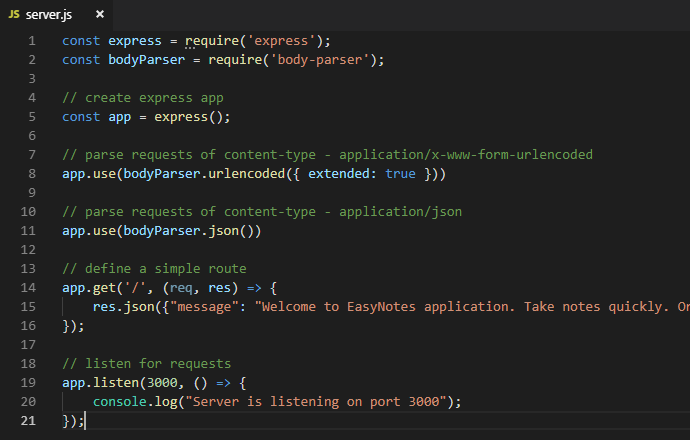


Our application folder now has a package.json file and a node\_modules folder

# 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



**First**,

We import express and body-parser modules.

[Express](https://www.npmjs.com/package/express), as you know, is a web framework that we’ll be using for building the REST APIs, and

[body-parser](https://www.npmjs.com/package/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](http://expressjs.com/en/guide/writing-middleware.html) 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.

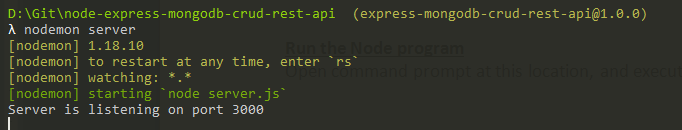
**Finally**,

We listen on port 3000 for incoming connections.

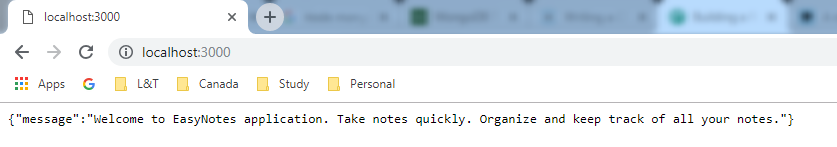
All right! Let’s now run the server and go to [http://localhost:3000](http://localhost:3000/) to access the route we just defined.

**Run the Node program**

Open command prompt at this location, and execute below command



Testing on browser

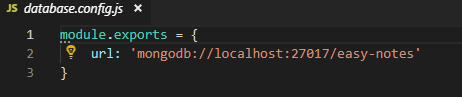
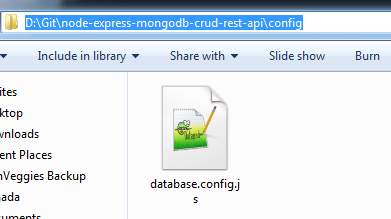


# Configuring and Connecting to the database

Let’s keep all the configurations for the app in a separate folder.

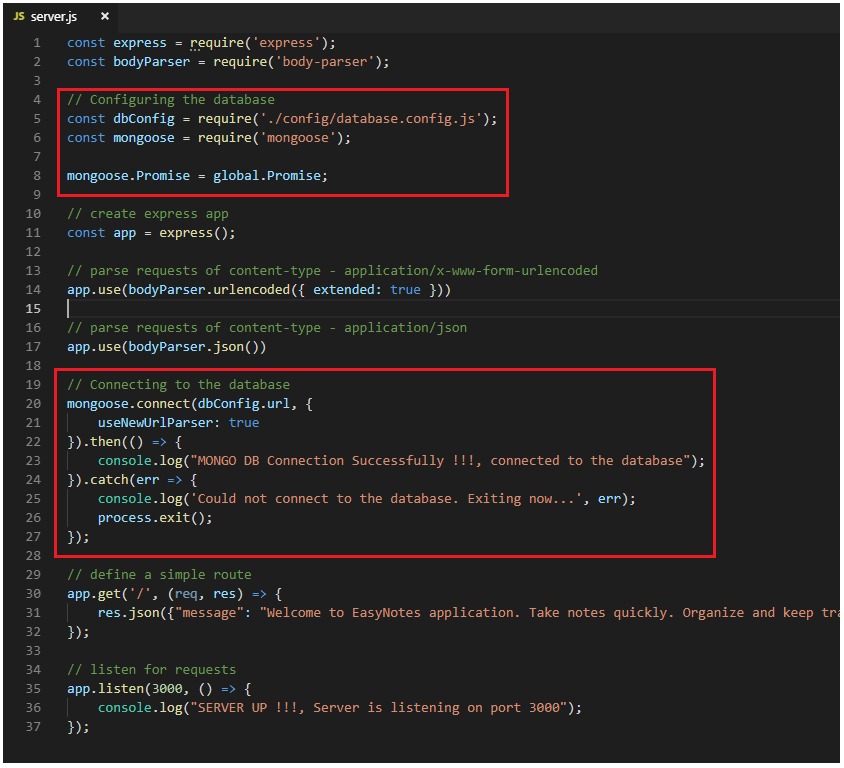
Let’s create a new folder config in the root folder of our application for keeping all the configurations

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

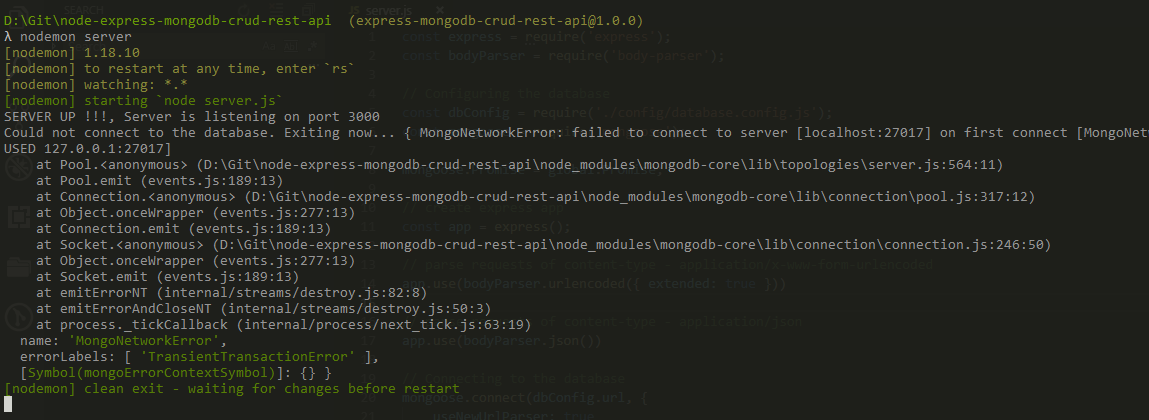
 

**We’ll now import the above database configuration in server.js and connect to the database using mongoose.**

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

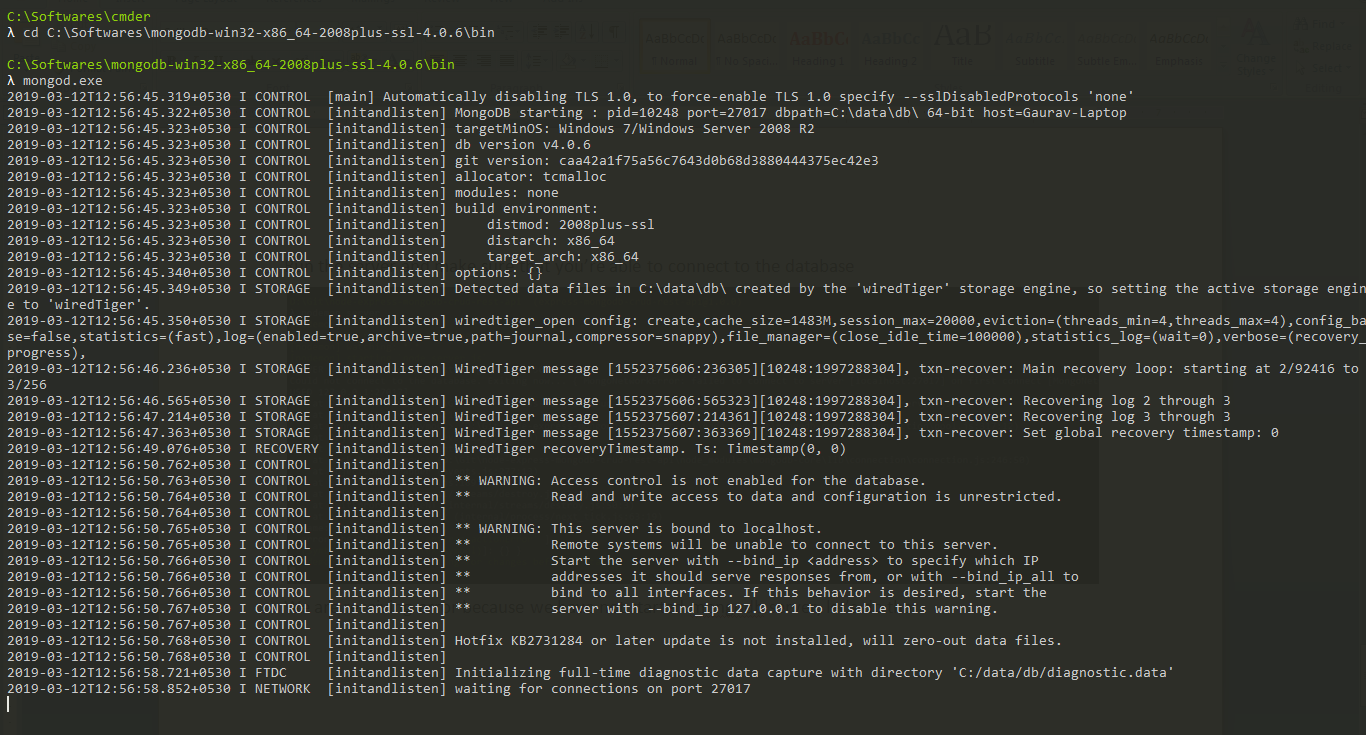


Run the server and make sure that you’re able to connect to the database

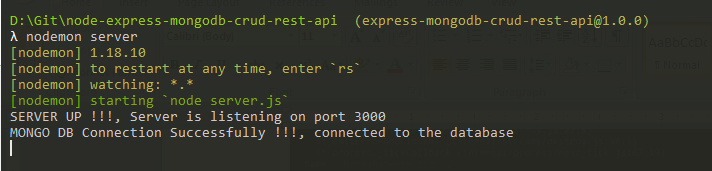


We are receiving error because we have not started mongodb server, let’s do that

Start Mongo Demon



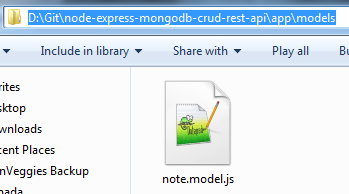
Now again try to connect with the node application



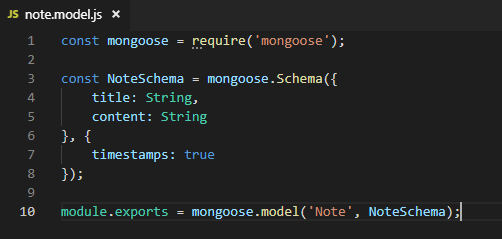
# Defining the Note model in Mongoose

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



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



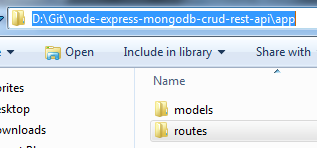
The Note model is very simple. It contains a title and a content field.

We have also added a [timestamps](http://mongoosejs.com/docs/guide.html#timestamps) option to the schema.

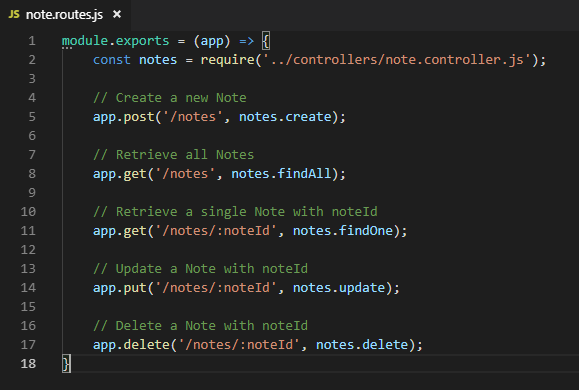
**Mongoose uses this option to automatically add two new fields - createdAt and updatedAt to the schema**.

# Defining Routes using Express

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



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

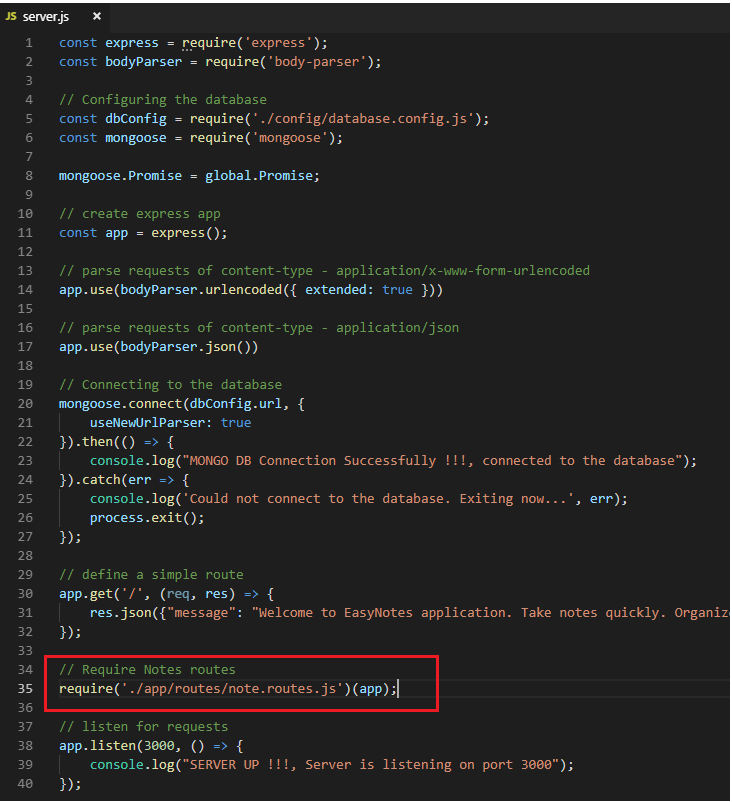


Note that We have added a require statement for note.controller.js file.

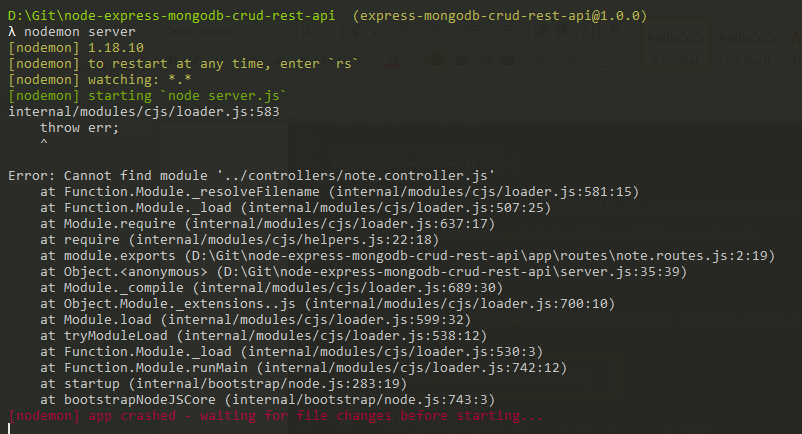
The controller will contain methods for handling all the **CRUD operations**.

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.



If you run the server now, you’ll get the following error

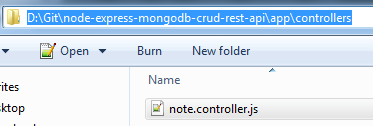


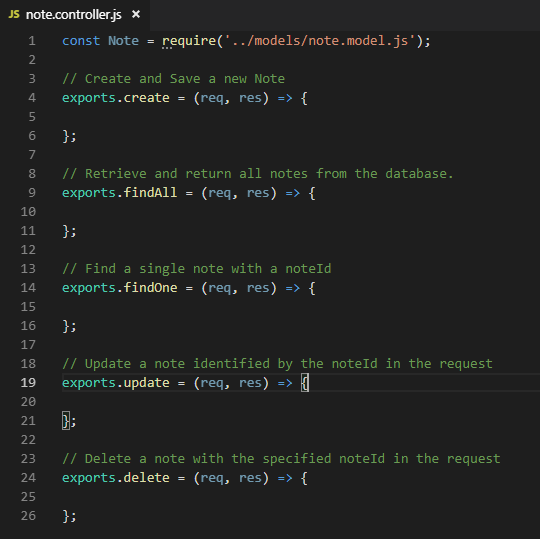
This is because we haven’t defined the controller yet. Let’s do that now.

# Writing the Controller functions

Create a new folder called controllers inside the app folder,

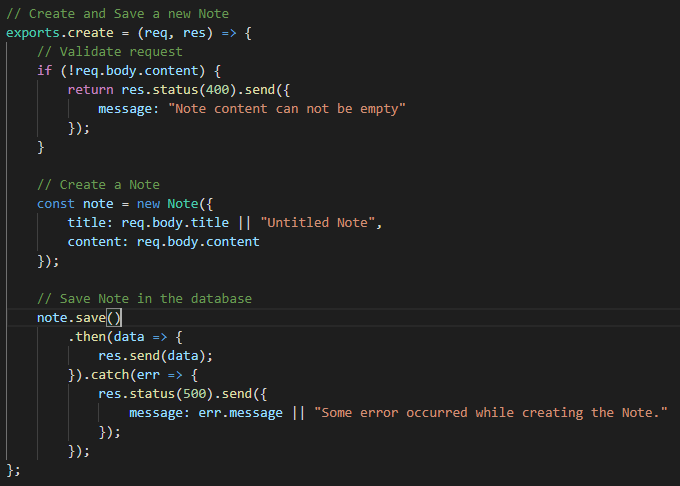
then create a new file called note.controller.js inside app/controllers folder with the following contents



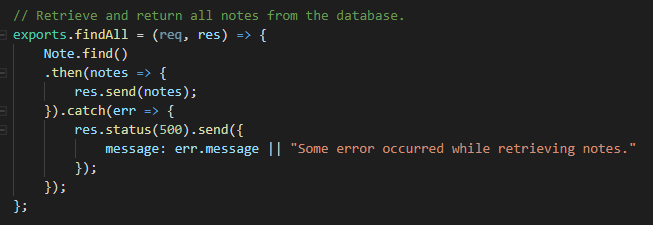


Let’s now look at the implementation of the above controller functions one by one

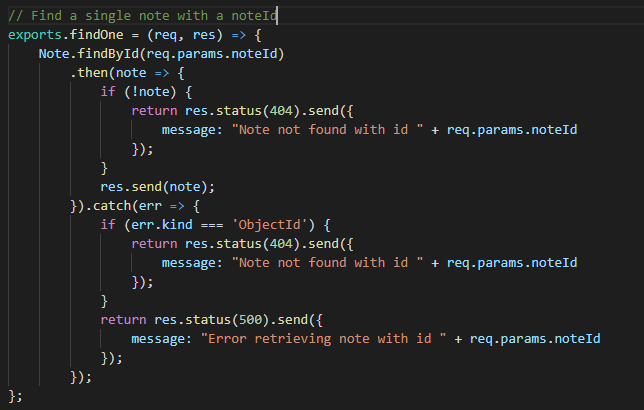
## Creating a new Note



## Retrieving all Notes



## Retrieving a single Note

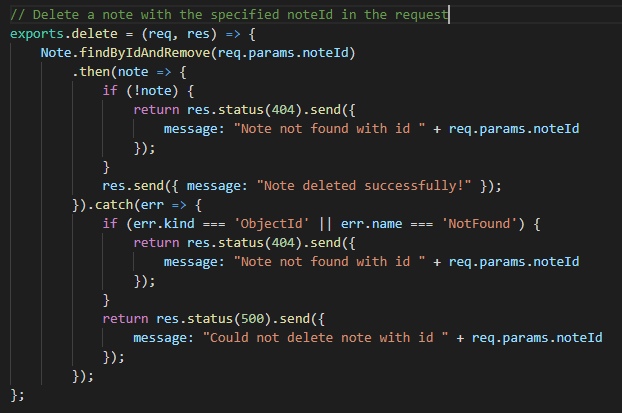


## Updating a Note



The {new: true} option in the [findByIdAndUpdate()](http://mongoosejs.com/docs/api.html" \l "findbyidandupdate_findByIdAndUpdate) method is used to return the modified document to the then() function instead of the original.

## Deleting a Note

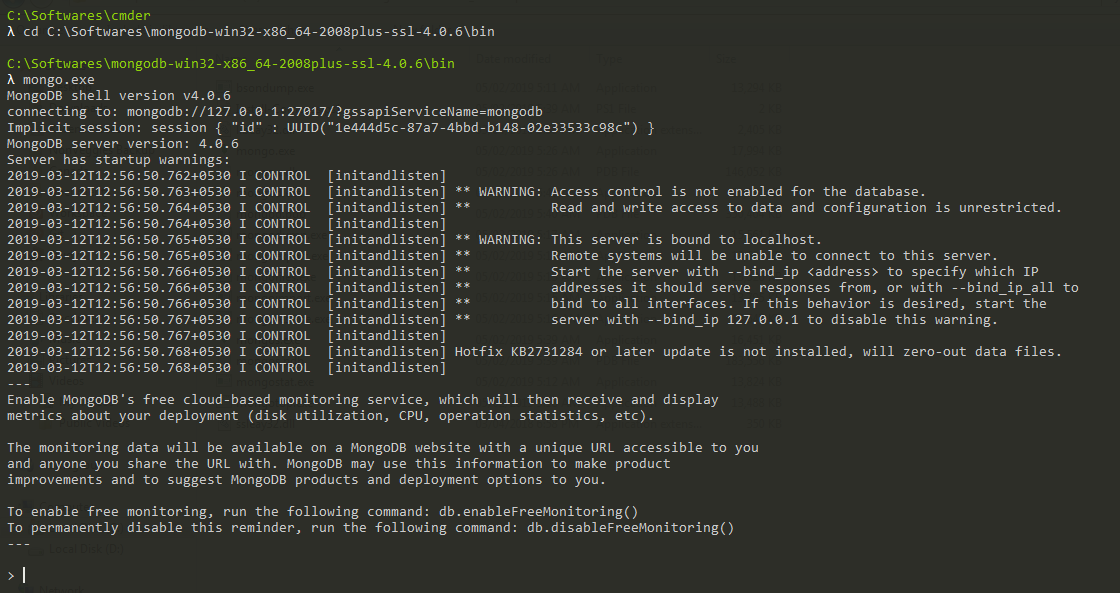


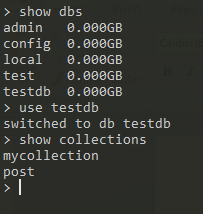
You can check out the documentation of all the methods that we used in the above APIs on Mongoose’s official documentation -

* [Mongoose save()](http://mongoosejs.com/docs/api.html#document_Document-save)
* [Mongoose find()](http://mongoosejs.com/docs/api.html#find_find)
* [Mongoose findById()](http://mongoosejs.com/docs/api.html#findbyid_findById)
* [Mongoose findByIdAndUpdate()](http://mongoosejs.com/docs/api.html#findbyidandupdate_findByIdAndUpdate)
* [Mongoose findByIdAndRemove()](http://mongoosejs.com/docs/api.html#findbyidandremove_findByIdAndRemove)

# Testing our APIs

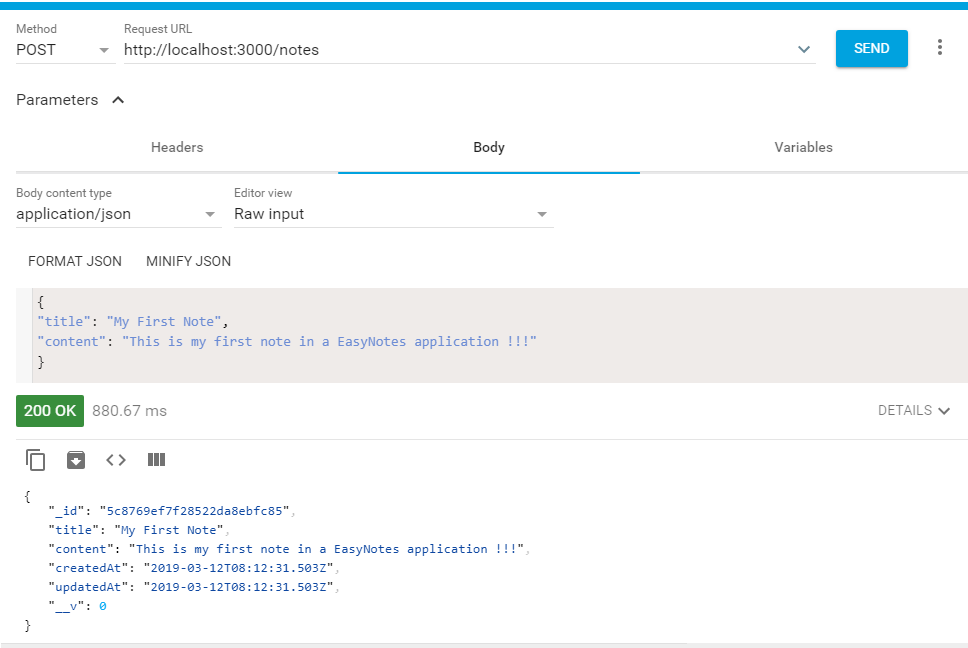
First test that we don’t have any documents and collections in DB

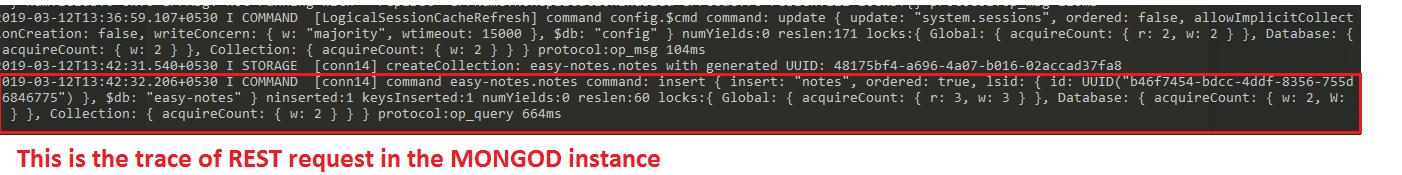




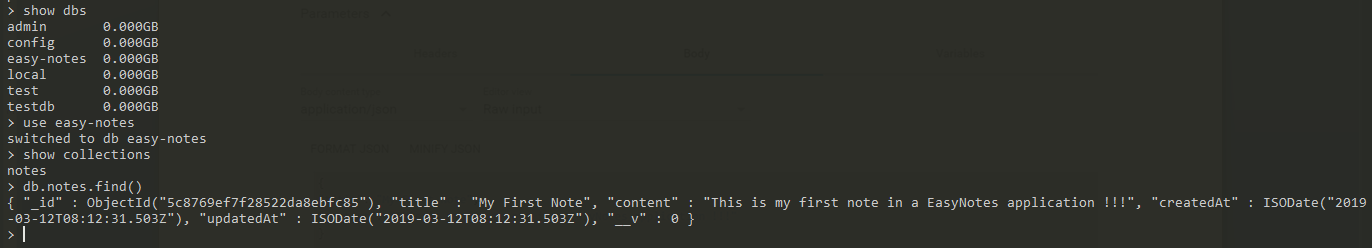
## ****Creating a new Note using****POST /notes****API****

On REST client



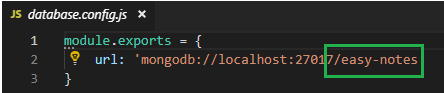


Verification from MongoClient



This same response is sent back to the rest client as response, in the above REST client snapshot.

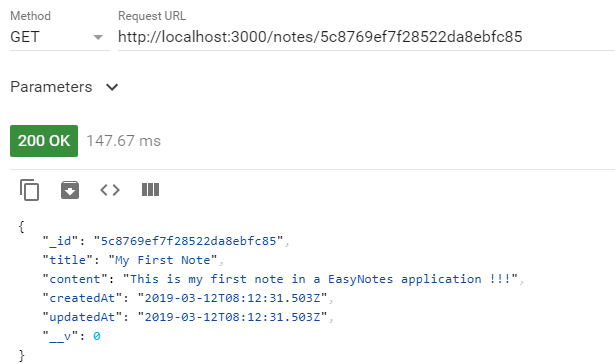
Here DB name “easy-notes” is fetched from database.config.js configuration file.



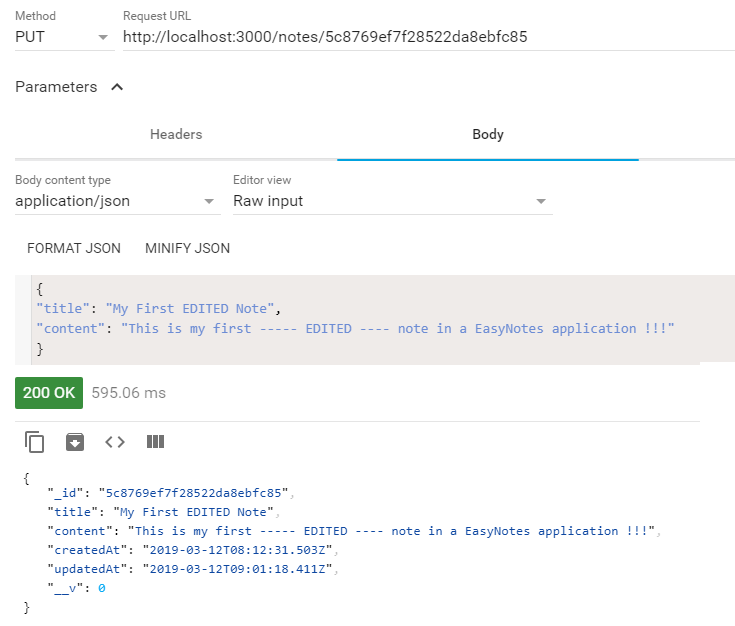
## Retrieving all Notes using GET /notes API



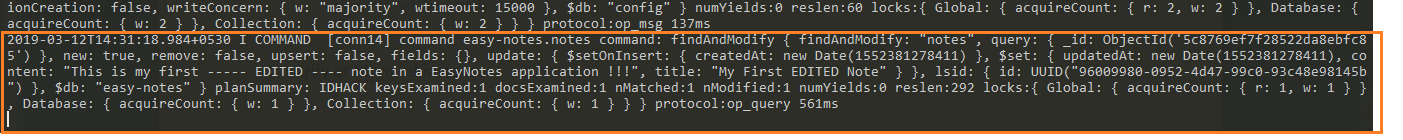
## Retrieving a single Note using GET /notes/:noteId API



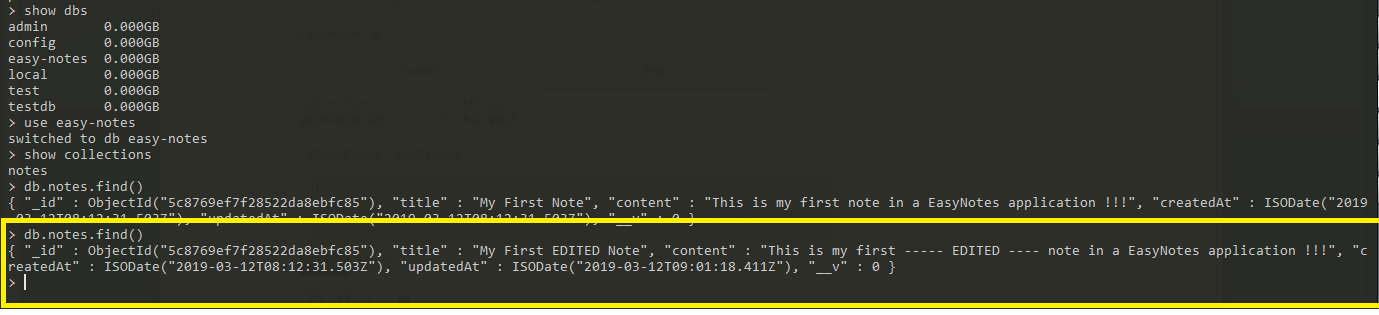
## Updating a Note using PUT /notes/:noteId API



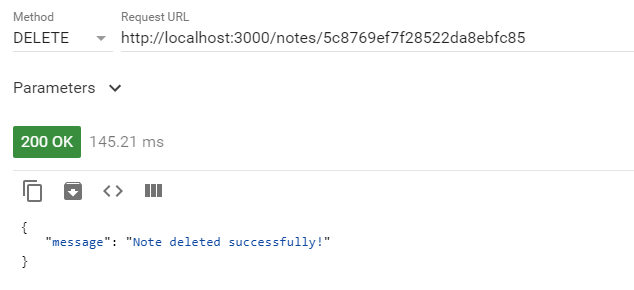
Verify at Mongod instance



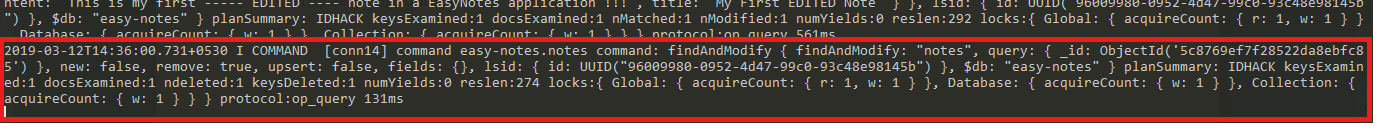
Verify at mongo Client



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



Verify at mongod instance

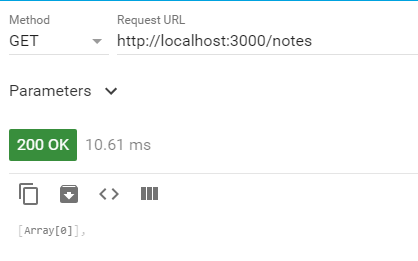


Verify at mongo client



No result is returned.

Query from REST API



# REFERENCE

<https://www.callicoder.com/node-js-express-mongodb-restful-crud-api-tutorial/>