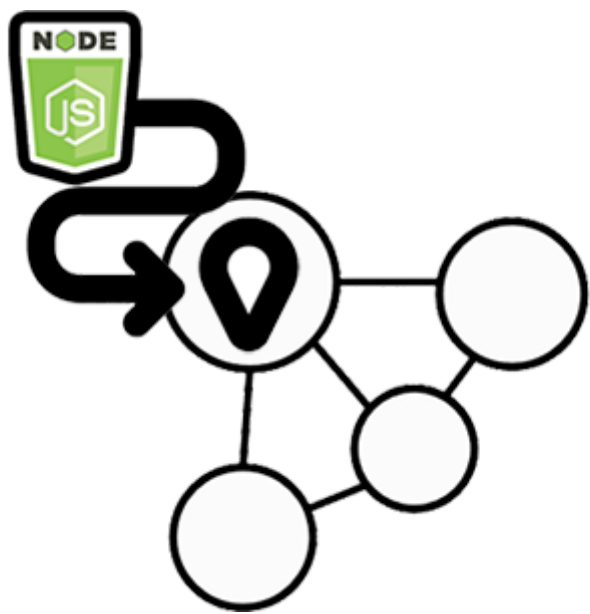


# How To Add Routes and Models To Node Rest API



At the conclusion of our last tutorial which looks at working with

Models in Mongoose, we now have a REST api which allows us to create new users and authenticate users. In a prior tutorial we also built out a restful resource for Games. In other words, we were able to create, read, update, and delete Games using the Node powered api. Now, we'll once again go over the process of how routes files and model files work together in an express app.


Let's start at index.js and see what we have so far.

```
1 app.use(express.json());
2 app.use('/api/users', users);
3 app.use('/api/auth', auth);
```

Notice the second arguments of `users` and `auth`. Those correspond to the `/routes/users.js` and `/routes/auth.js` `routes` files. Now we are going to add a way to allow users to add a new Platform for which will allow users to set up various game platforms such as Nintendo Switch, Xbox One, or Sony Playstation. Let's see how to set this up.






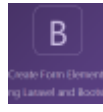






## 1. Set Up index.js

First we can configure the index.js file for a new route path / routes file association.



Get a free VPS with DigitalOcean!  
You will receive \$100 in DigitalOcean credit when you create a new account!

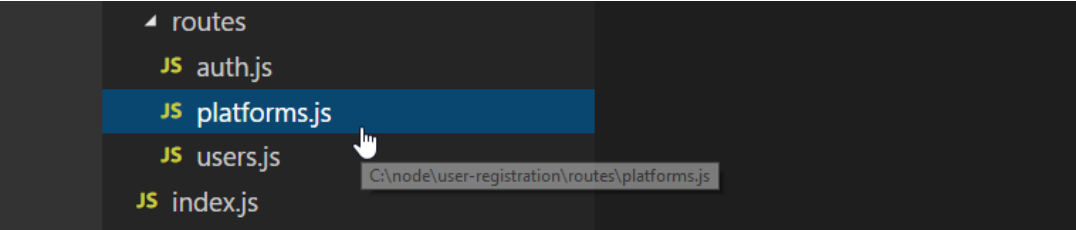
Advertise Here

- [Python Dictionaries](#)
- [What are Getters and Setters?](#)
- [Laravel String Helpers](#)
- [Laravel Vue Component Example](#)
- [Getting Your Database Set Up For Use With Laravel](#)
- [Create Form Elements Using Laravel and Bootstrap](#)
- [Setting Up A Database With Seeding](#)
- [How Does The Filter Function Work In Underscore JS?](#)
- [How To Filter Via Query Strings](#)
- [Autoloading For Code Organization](#)
- [C# Classes For A CRM Application](#)
- [Adding Game Reviews With](#)

```
1 const config = require('config');
2 const Joi = require('joi');
3 const mongoose = require('mongoose');
4 const platforms = require('./routes/platforms');
5 const users = require('./routes/users');
6 const auth = require('./routes/auth');
7 const express = require('express');
8 const app = express();
9
10 if (!config.get('PrivateKey')) {
11   console.error('FATAL ERROR: PrivateKey is not defined');
12   process.exit(1);
13 }
14
15 mongoose.connect('mongodb://localhost/mongo-game', {
16   useNewUrlParser: true,
17   useUnifiedTopology: true
18 }).then(() => console.log('Now connected to MongoDB'))
19   .catch(err => console.error('Something went wrong'));
20
21 app.use(express.json());
22 app.use('/api/platforms', platforms);
23 app.use('/api/users', users);
24 app.use('/api/auth', auth);
25
26 const port = process.env.PORT || 4000;
27 app.listen(port, () => console.log(`Listening on port ${port}`));
```

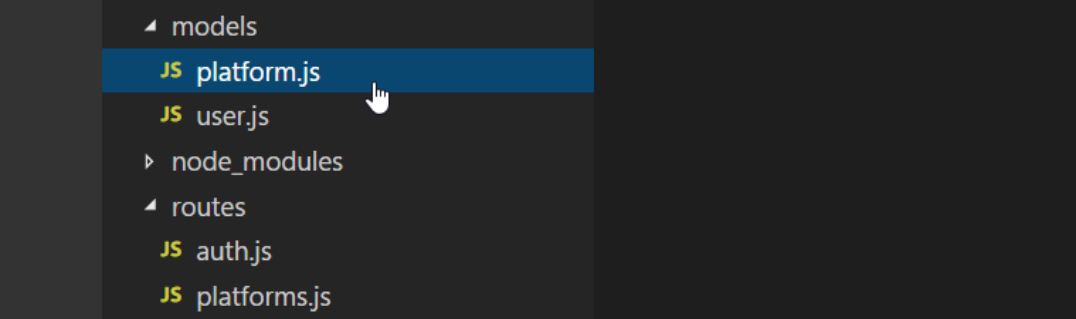
## 2. Set up platforms.js Routes file

Now we need to add a new file in our routes directory like so. We'll call it **platforms.js**.



## 3. Set up platform.js Model file

We also need a model for the Platform, we can add a platform.js model file to the `/models` directory.



## Building Out A Model File

### Eloquent Relationships



[Angular Pipes And Data Binding](#)



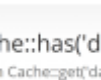
[Angular Structural](#)

### Directives



[How Do Linux Permissions](#)

### Work?



[Laravel Cache Tutorial](#)



[Build Your Own PCRE Regex](#)

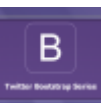
### Tester With PHP



[Twitter Bootstrap Modal Tutorial](#)



[Mithril JavaScript Tutorial](#)



[What Is Twitter Bootstrap?](#)



[PHP MySQL CRUD Tutorial](#)



[Vue Sibling Component Communication](#)



[jQuery Autocomplete As You Type](#)



[ES6 Rest Parameters and Spread Operators](#)



[Traverse the DOM With jQuery](#)



[Multi-Column Layout With CSS](#)



[Linux Files And Directories](#)

### Commands



[The Most Popular Content](#)

First up, we'll require Joi and Mongoose, and begin building a new schema. The Platform will be a string that is required with a minimum length of 5 and a maximum length of 50.

```
1 const Joi = require('joi');
2 const mongoose = require('mongoose');
3
4 const platformSchema = new mongoose.Schema({
5   name: {
6     type: String,
7     required: true,
8     minlength: 5,
9     maxlength: 50
10  }
11 });
```

Then, using our Schema, we can easily compile a model from it like so.

```
1 const Platform = mongoose.model('Platform', platformS
```

Now, we can build a very simple validation function which specifies a 3 character minimum for the Platform.

```
1 function validatePlatform(platform) {
2   const schema = {
3     name: Joi.string().min(3).required()
4   };
5
6   return Joi.validate(platform, schema);
7 }
```

Lastly, we'll export the platformSchema, the Platform model, and the validatePlatformfunction. These lines are highlighted in the context of our full platform.js file here.



Manipulation Methods in jQuery



VegiThemes  
Twitter Bootstrap Themes



Using Operators in PHP



The Top 100 Most Commonly Used WordPress Functions



Linux Filesystem Hierarchy Explained



CSS Grid Tutorial



Underscore JS sortBy Function



What Are Scalable Vector Graphics?



Install Laravel Homestead on Windows



Build A Link Sharing Website With Laravel



How To Test Your Laravel Application



WordPress Links and Images



PHP Include Vs Require



Creating C# Class Properties And Tests



What is WordPress?



Composing React Components



C# Relationships Between Classes

```
1 const Joi = require('joi');
2 const mongoose = require('mongoose');
3
4 const platformSchema = new mongoose.Schema({
5   name: {
6     type: String,
7     required: true,
8     minlength: 5,
9     maxlength: 50
10  }
11 });
12
13 const Platform = mongoose.model('Platform', platformSchema);
14
15 function validatePlatform(platform) {
16   const schema = {
17     name: Joi.string().min(3).required()
18   };
19
20   return Joi.validate(platform, schema);
21 }
22
23 exports.platformSchema = platformSchema;
24 exports.Platform = Platform;
25 exports.validate = validatePlatform;
```

# Building Out A Routes File

In this case, we’re building the routes file last because we are actually depending on code that was written in the model file for use in our routes file. In this file we’ll need access to the Platform model, the validate function, mongoose, express, and the express router. We set all of that up first.

```
1 const {Platform, validate} = require('../models/platform');
2 const mongoose = require('mongoose');
3 const express = require('express');
4 const router = express.Router();
```

# Configure POST Request To Add a Platform

We don’t have any Platforms in our system yet, so lets first build a way to add some using our REST api. Here we validate a request, and if there are no errors, persist the platform to Mongoddb.

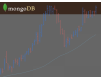
## Process HTML Forms With



[PHP](#)



[Douglas Crockford The Good Parts Examples](#)



[Node MongoDB Native](#)



[PHP Booleans and Constants](#)

## Tutorial



[Twitter Bootstrap 12 Column Grid](#)



[Angular Styles Vs StyleUrls](#)



[Twitter Bootstrap Classes](#)



[Command Types In Linux](#)



[CSS Transitions and](#)

## Transformations



[How To Fix The N+1 Problem](#)



[How To Compare Two Arrays of](#)

## Data and Calculate Position Differences



[How To Add Routes and](#)

## Models To Node Rest API



[Introduction To Laravel](#)

## Controllers



[The Top 17 Most Popular PHP](#)

## Array Functions



[Dependency Inversion](#)

## Principle



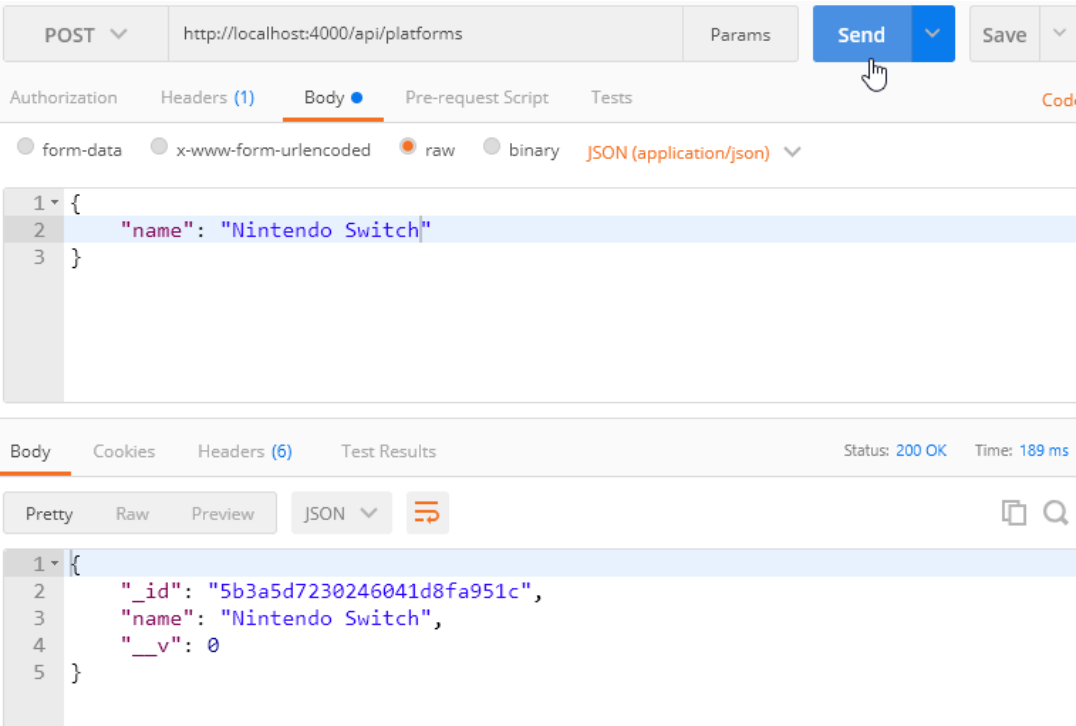
[Laravel Migration Generator](#)



```
1 router.post('/', async (req, res) => {
2   const { error } = validate(req.body);
3   if (error) {
4     return res.status(400).send(error.details[0].message);
5   }
6
7   let platform = new Platform({ name: req.body.name });
8   platform = await platform.save();
9
10  res.send(platform);
11 });
12
13 module.exports = router;
```

Important! Don't forget that `module.exports = router;` code, otherwise you will suffer some error such as *throw new TypeError('Router.use() requires a middleware function but got a ' + gettype(fn))*

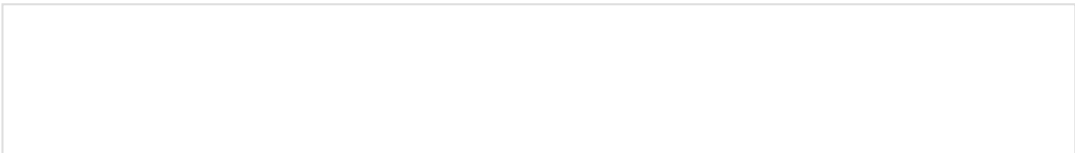
Ok! I think we can try adding a new Platform to the system by sending a POST request to our new endpoint of `http://localhost:4000/api/platforms`. Here we will add your favorite gaming platform and mine, the Nintendo Switch. Boot up the server by running *nodemon index.js* and we can test it out.

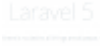












In the background we can add several more platforms as well so that we have a few we can work with.

## Configure a GET request to view all Platforms

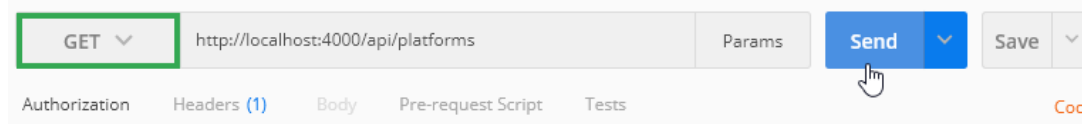
Wouldn't you like to know the other platforms we added to the system? Of course you would. This is why we will add a GET request endpoint to our api so we can now view all platforms currently in MongoDB. Check it out.



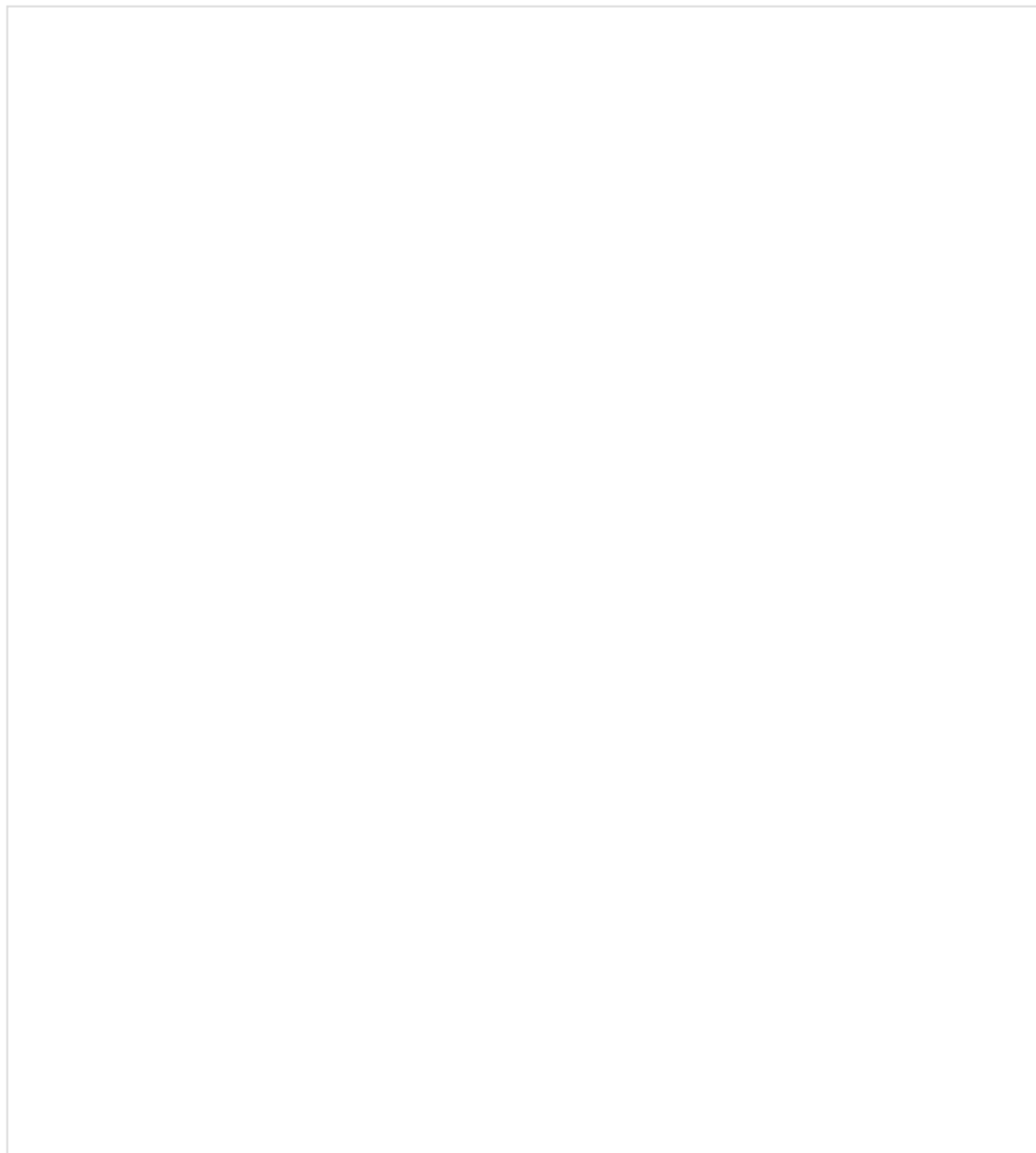
- [Introduction To Laravel 5](#)
- [What is a View Composer in Laravel?](#)
- [C# Classes And Objects](#)
- [Setting Permissions With Policy Objects](#)
- [Encapsulation For Hackers](#)
- [Display Activity Feed In The Browser](#)
- [VueJS Image Upload](#)
- [WordPress Post Formats](#)
- [Check Authorization With Policies Before Delete Function](#)
- [How To Set Up Form Submission In Laravel](#)
- [The Ultimate Guide to PHP Functions](#)

```
1 const { Platform, validate } = require('../models/platform');
2 const mongoose = require('mongoose');
3 const express = require('express');
4 const router = express.Router();
5
6 router.post('/', async (req, res) => {
7   const { error } = validate(req.body);
8   if (error) {
9     return res.status(400).send(error.details[0].message);
10  }
11
12  let platform = new Platform({ name: req.body.name });
13  platform = await platform.save();
14
15  res.send(platform);
16 });
17
18 router.get('/', async (req, res) => {
19   const platforms = await Platform.find().sort('name');
20   res.send(platforms);
21 });
22
23 module.exports = router;
```

Simple! Now, make a GET request in Postman.

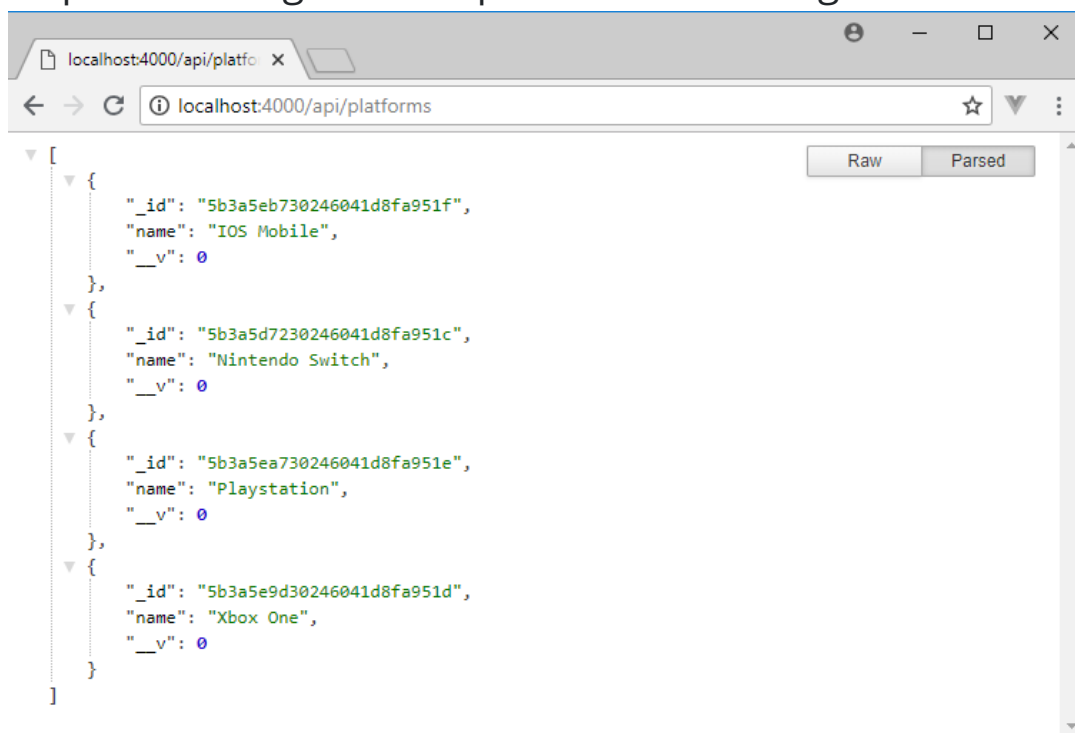


We get this pretty sweet JSON collection of Video Game Platforms.



```
1 [
2   {
3     "_id": "5b3a5eb730246041d8fa951f",
4     "name": "IOS Mobile",
5     "__v": 0
6   },
7   {
8     "_id": "5b3a5d7230246041d8fa951c",
9     "name": "Nintendo Switch",
10    "__v": 0
11  },
12  {
13    "_id": "5b3a5ea730246041d8fa951e",
14    "name": "Playstation",
15    "__v": 0
16  },
17  {
18    "_id": "5b3a5e9d30246041d8fa951d",
19    "name": "Xbox One",
20    "__v": 0
21  }
22 ]
```

You may also make a simple GET request using a web browser to the `http://localhost:4000/api/platforms` endpoint. Just paste in the endpoint and hit enter! Why? Because any time you are trying to view something in a web browser, the browser is simply making a GET request to the given URL provided. So here goes.



## Configure a PUT request endpoint to modify Platforms

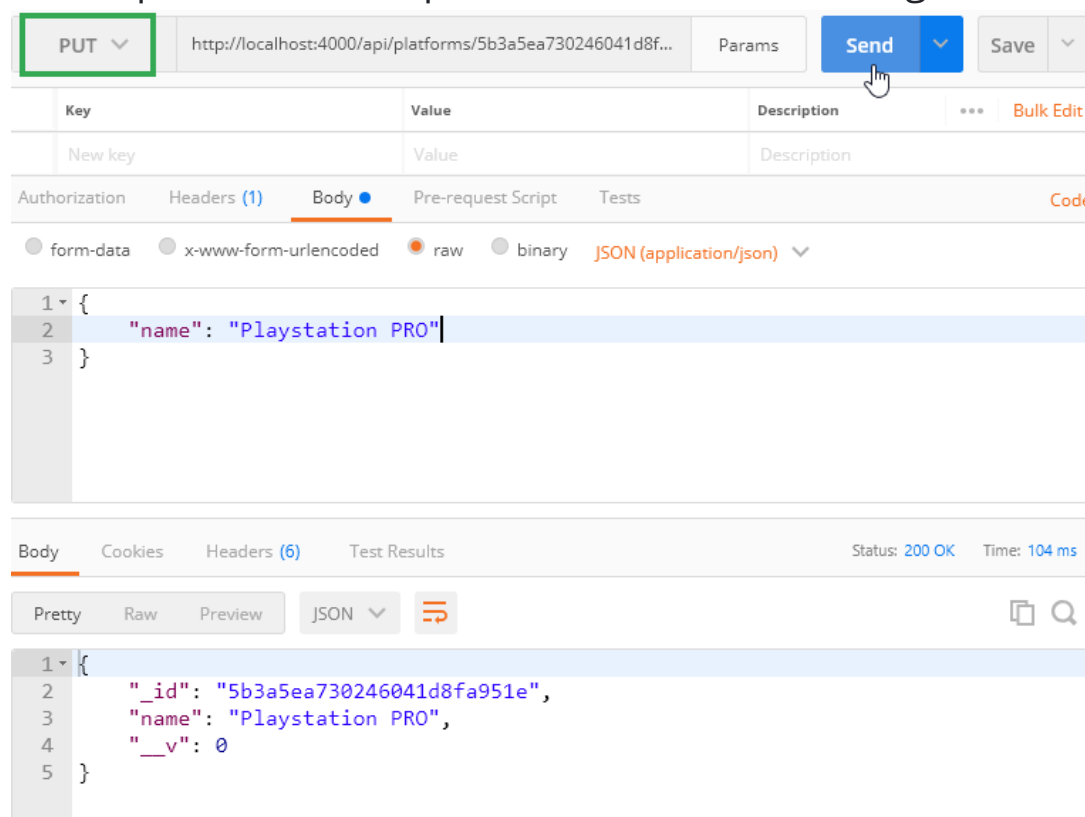
If you want to be able to send a PUT request to the API in order to make an update to a resource, here is how we can do it. We'll make use of the handy [\*\*findByIdAndUpdate\*\*](#) method.

```

1 router.put('/:id', async (req, res) => {
2   const { error } = validate(req.body);
3   if (error) {
4     return res.status(400).send(error.details[0].message);
5   }
6
7   const platform = await Platform.findByIdAndUpdate(
8     req.params.id, req.body, {
9     new: true
10    });
11
12   if (!platform) {
13     return res.status(404).send('That platform ID was not found');
14   }
15   res.send(platform);
16 });

```

Now, we want to change the name of the Playstation platform to **Playstation PRO**. First, we determine the id of the Platform we want to update. In this case, it is **5b3a5ea730246041d8fa951e**. Therefore, we send a PUT request using postman to `http://localhost:4000/api/platforms/5b3a5ea730246041d8fa951e` while also setting a JSON object in the body of the request with the updated name value. Bazinga!



## Configure a DELETE request endpoint to delete Platforms

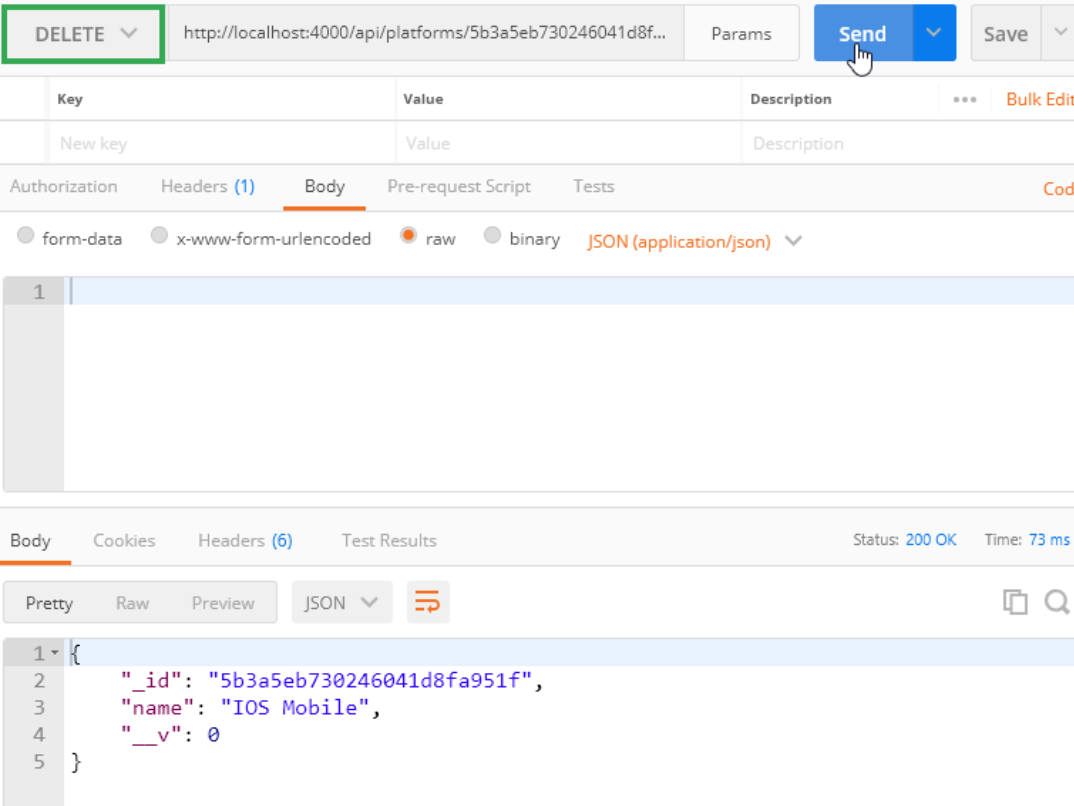
Now we want to allow for the ability to delete a Platform from the system. So you're probably seeing a theme here. For each resource, it is pretty easy to set up the CRUD functions. Here is that DELETE option we want for our Platforms resource. Here is our code to handle the



DELETE request which completes all of the crud operations for this resource. Note the handy use of [findByIdAndRemove](#).

```
1 router.delete('/:id', async (req, res) => {
2   const platform = await Platform.findByIdAndRemove(
3
4   if (!platform) {
5     return res.status(404).send('That platform ID was
6   }
7
8   res.send(platform);
9 });
```

So what should we delete? Let’s delete the Mobile IOS platform. First, we determine it’s id of **5b3a5eb730246041d8fa951f**. We’ll need that to send a correct DELETE request with Postman. Here we go.



## How To Add Routes and Models To Node Rest API Summary

To be fair, a lot of what we did here is almost the same as what we covered in the [express rest api tutorial](#). In that first one we dealt with Games. Here we are working with Platforms those games can be played on. The take away however is getting our thinking about the process of imagining an endpoint we would like to have, creating a the app.use() statement in the index.js file, setting up a dedicated Model file, and finally, setting up the dedicated routes file. With this we understand how each of the files export and require and talk to each other to make the api work.

←

Information Expert Principle Applied To Mongoose Models

|

JSON Web Token Authentication With Node.js

→