With the release of [Express 4.0](http://expressjs.com/4x/api.html) just a few days ago, lots of our Node apps will have some changes in how they handle routing. With the changes in the [Express Router](http://expressjs.com/4x/api.html#router), we have more flexibility in how we can define the routes for our applications.

Today we’ll be looking at creating a RESTful API using Node, [Express 4 and its Router](https://scotch.io/tutorials/javascript/learn-to-use-the-new-router-in-expressjs-4), and Mongoose to interact with a MongoDB instance. We will also be testing our API using [Postman](https://chrome.google.com/webstore/detail/postman-rest-client-packa/fhbjgbiflinjbdggehcddcbncdddomop) in Chrome.

Let’s look at the API we want to build and what it can do.

[**Our Application**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#our-application)

We are going to build an API that will:

* Handle CRUD for an item (we’re going to use bears)
* Have a standard URL (http://example.com/api/bears andhttp://example.com/api/bears/:bear\_id)
* Use the proper HTTP verbs to make it RESTful (GET,POST, PUT, and DELETE)
* Return JSON data
* Log all requests to the console

All of this is pretty standard for [RESTful APIs](https://scotch.io/bar-talk/designing-a-restful-web-api). Feel free to switch out bears for anything that you will want to build for your application (users, superheroes, beers, etc).

Make sure you have [Node](http://nodejs.org/) installed and let’s get to it!

[**Getting Started**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#getting-started)

Let’s look at all the files we will need to create our API. We will need to **define our Node packages**, **start our server using Express**, **define our model**, **declare our routes using Express**, and last but not least, **test our API**.

Here is our file structure. We won’t need many files and we’ll keep this very simple for demonstration purposes. When moving to a production or larger application, you’ll want to separate things out into a good structure (like having your routes in their own file).

- app/

----- models/

---------- bear.js // our bear model

- node\_modules/ // created by npm. holds our dependencies/packages

- package.json // define all our node app and dependencies

- server.js // configure our application and create routes

**Defining our Node Packages**package.json

As with all of our Node projects, we will define the packages we need in package.json. Go ahead and create that file with these packages.

// package.json

{

"name": "node-api",

"main": "server.js",

"dependencies": {

"express": "~4.0.0",

"mongoose": "~3.6.13",

"body-parser": "~1.0.1"

}

}

What do these packages do? express is the Node framework. mongoose is the ORM we will use to communicate with our MongoDB database. body-parser will let us pull POST content from our HTTP request so that we can do things like create a bear.

**Installing Our Node Packages**

This might be the easiest step. Go into the command line in the root of your application and type:

$ npm install

npm will now pull in all the packages defined into anode\_modules folder in our project.

npm is Node’s package manager that will bring in all the packages we defined in package.json. Simple and easy. Now that we have our packages, let’s go ahead and use them when we set up our API.

We’ll be looking to our server.js file to setup our app since that’s the main file we declared in package.json.

**Setting Up Our Server**server.js

Node will look here when starting the application so that it will know how we want to configure our application and API.

We will start with the bear (get it?) essentials necessary to start up our application. We’ll keep this code clean and commented well so we understand exactly what’s going on every step of the way.

// server.js

// BASE SETUP

// =============================================================================

// call the packages we need

var express = require('express'); // call express

var app = express(); // define our app using express

var bodyParser = require('body-parser');

// configure app to use bodyParser()

// this will let us get the data from a POST

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

app.use(bodyParser.json());

var port = process.env.PORT || 8080; // set our port

// ROUTES FOR OUR API

// =============================================================================

var router = express.Router(); // get an instance of the express Router

// test route to make sure everything is working (accessed at GET <http://localhost:8080/api>)

router.get('/', function(req, res) {

res.json({ message: 'hooray! welcome to our api!' });

});

// more routes for our API will happen here

// REGISTER OUR ROUTES -------------------------------

// all of our routes will be prefixed with /api

app.use('/api', router);

// START THE SERVER

// =============================================================================

app.listen(port);

console.log('Magic happens on port ' + port);

Wow we did a lot there! It’s all very simple though so let’s walk through it a bit.

**Base Setup** In our base setup, we pull in all the packages we pulled in using npm. We’ll grab express, define our app, get bodyParser and configure our app to use it. We can also set the port for our application.

**Routes for Our API** This section will hold all of our routes. The structure for using the Express Router let’s us pull in an instance of the router. We can then **define routes** and then **apply those routes** to a root URL (in this case, API).

**Start our Server** We’ll have our express app listen to the port we defined earlier. Then our application will be live and we can test it!

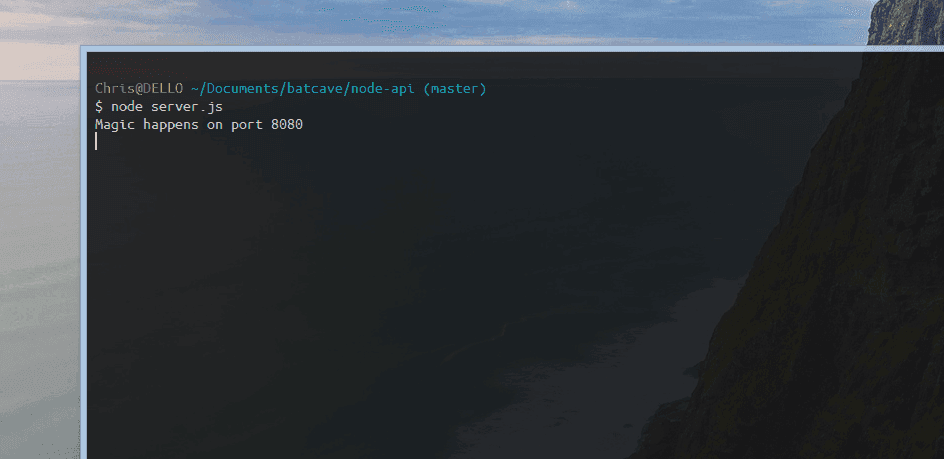
[**Starting Our Server and Testing**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#starting-our-server-and-testing)

Let’s make sure that everything is working up to this point. We will start our Node app and then send a request to the one route we defined to make sure we get a response.

Let’s start our server. From the command line, type:

$ node server.js

You should see your Node app start up and Express will create a server.

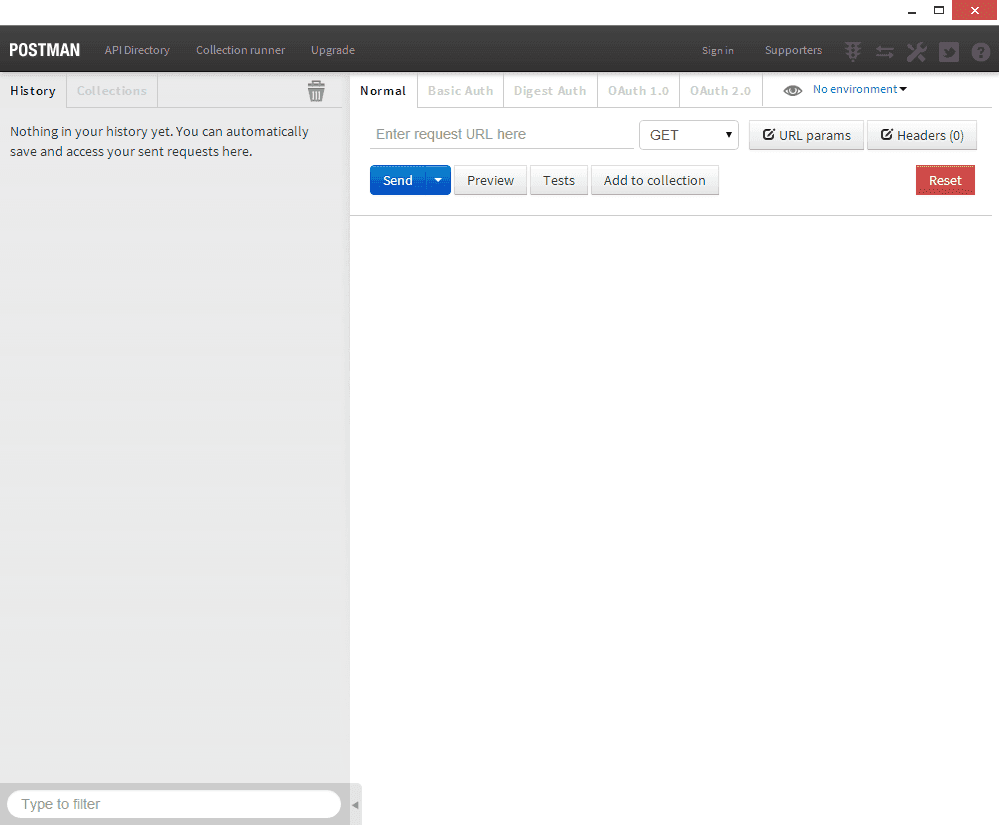
[](https://cask.scotch.io/2014/04/node-api-start-server.png)

Now that we know our application is up and running, let’s test it.

**Testing Our API Using Postman**

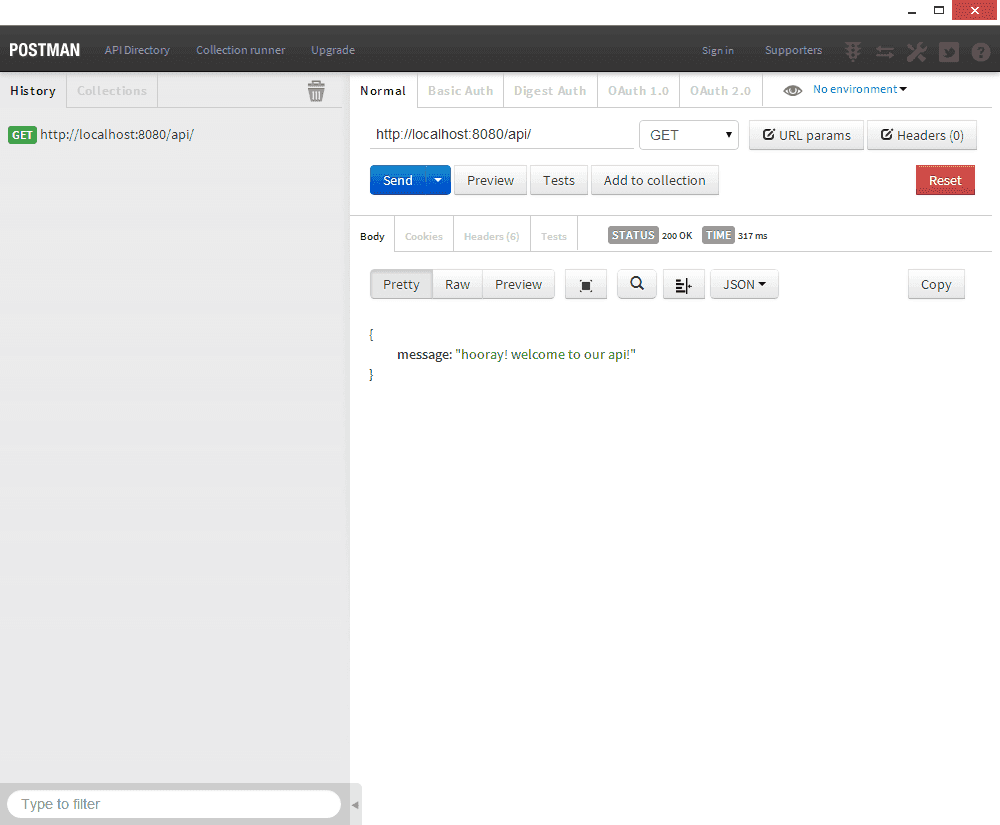
Postman will help us test our API. It will basically send HTTP requests to a URL of our choosing. We can even pass in parameters (which we will soon) and authentication (which we won’t need for this tutorial).

Open up Postman and let’s walk through how to use it.

[](https://cask.scotch.io/2014/04/postman-rest-client-node-api.png)

All you have to do is **enter your request URL**, **select an HTTP verb**, and click **Send**. Simple enough right?

Here’s the moment we’ve been waiting for. Does our application work the way we configured it? Enterhttp://localhost:8080/api into the URL. GET is what we want since we just want to get data. Now click **Send**.

[](https://cask.scotch.io/2014/04/node-api-postman-test.png)

Sweet! We got back exactly what we wanted. Now we know we can serve information to requests. Let’s wire up our database so we can start performing CRUD operations on some bears.

[**Database and Bear Model**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#database-and-bear-model)

We’ll keep this short and sweet so that we can get to the fun part of building the API routes. All we need to do is create a MongoDB database and have our application connect to it. We will also need to create a bear mongoose model so we can use mongoose to interact with our database.

**Creating Our Database and Connecting**

We will be using a database provided by [Modulus](http://modulus.io/). You can definitely create your own database and use it locally or use the awesome [Mongolab](https://mongolab.com/" \t "_blank). All you really need is a URI like below so that your application can connect.

Once you have your database created and have the URI to connect to, let’s add it to our application. Inserver.js in the Base Setup section, let’s add these two lines.

// server.js

// BASE SETUP

// =============================================================================

...

var mongoose = require('mongoose');

mongoose.connect('<mongodb://node:node>@novus.modulusmongo.net:27017/Iganiq8o'); // connect to our database

...

That will grab the mongoose package and connect to our remote database hosted by Modulus. Now that we are connected to our database, let’s create a mongoose model to handle our bears.

**Bear Model**app/models/bear.js

Since the model won’t be the focus of this tutorial, we’ll just create a model and provide our bears with a name field. That’s it. Let’s create that file and define the model.

// app/models/bear.js

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var BearSchema = new Schema({

name: String

});

module.exports = mongoose.model('Bear', BearSchema);

With that file created, let’s pull it into our server.js so that we can use it within our application. We’ll add one more line to that file.

// server.js

// BASE SETUP

// =============================================================================

...

var Bear = require('./app/models/bear');

...

Now our entire application is ready and wired up so we can start building out our routes. These routes will define our API and the main reason why this tutorial exists. Moving on!

[**Express Router and Routes**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#express-router-and-routes)

We will use an instance of the Express Router to handle all of our routes. Here is an overview of the routes we will require, what they will do, and the HTTP Verb used to access it.

| Route | HTTP Verb | Description |
| --- | --- | --- |
| /api/bears | GET | Get all the bears. |
| /api/bears | POST | Create a bear. |
| /api/bears/:bear\_id | GET | Get a single bear. |
| /api/bears/:bear\_id | PUT | Update a bear with new info. |
| /api/bears/:bear\_id | DELETE | Delete a bear. |

This will cover the basic routes needed for an API. This also keeps to a good format where we have kept the actions we need to execute (GET, POST, PUT, and DELETE) as HTTP verbs.

[**Route Middleware**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#route-middleware)

We’ve already defined our first route and seen it in action. The Express Router gives us a great deal of flexibility in definining our routes.

Let’s say that we **wanted something to happen every time a request was sent to our API**. For this example we are just going to console.log() a message. Let’s add that middleware now.

// server.js

...

// ROUTES FOR OUR API

// =============================================================================

var router = express.Router(); // get an instance of the express Router

// middleware to use for all requests

router.use(function(req, res, next) {

// do logging

console.log('Something is happening.');

next(); // make sure we go to the next routes and don't stop here

});

// test route to make sure everything is working (accessed at GET <http://localhost:8080/api>)

router.get('/', function(req, res) {

res.json({ message: 'hooray! welcome to our api!' });

});

// more routes for our API will happen here

// REGISTER OUR ROUTES -------------------------------

// all of our routes will be prefixed with /api

app.use('/api', router);

...

All we needed to do to declare that middleware was to use router.use(function()). The order of how we define the parts of our router is very important. They will run in the order that they are listed and thanks to the [changes in Express 4.0](https://scotch.io/bar-talk/expressjs-4-0-new-features-and-upgrading-from-3-0), we won’t have problems doing this like in Express 3.0. Everything will run in the correct order.

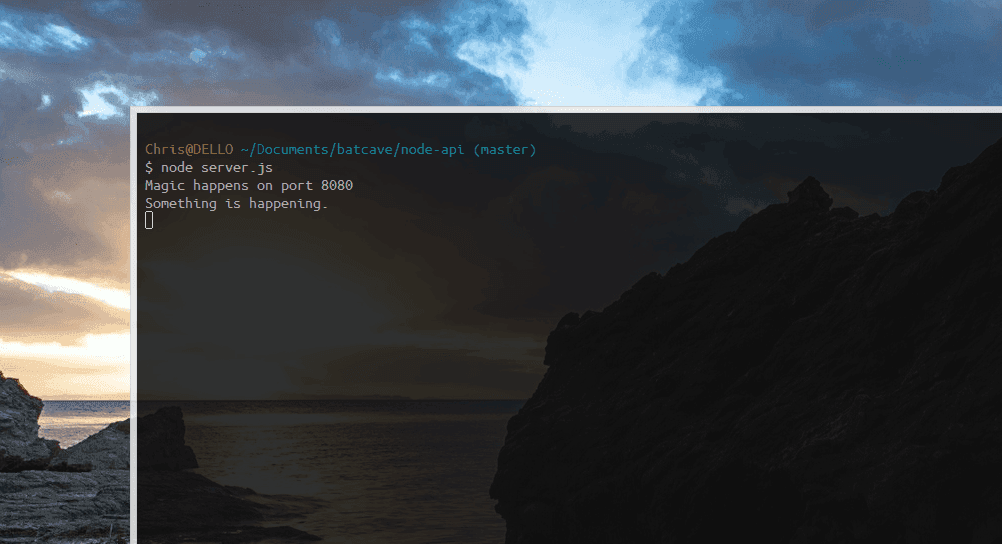
We are sending back information as **JSON data**. This is standard for an API and will help the people using our API to use our data.

We will also add next() to indicate to our application that it should continue to the other routes. This is important because our application would stop at this middleware without it.

**Middleware Uses** Using middleware like this can be very powerful. We can do validations to make sure that everything coming from a request is safe and sound. We can throw errors here in case something is wrong. We can do some extra logging for analytics or any statistics we’d like to keep. There are many possibilities here. Go wild.

**TESTING OUR MIDDLEWARE**

Now when we send a request to our application using Postman, Something is happening will be logged to our Node console (the command line).

[](https://cask.scotch.io/2014/04/node-api-route-middleware-express.png)

With middleware, we can do awesome things to requests coming into our API. We will probably want to make sure that the user is authenticated to access our API. We’ll go over that in a future article, but for now let’s just log something to the console with our middleware.

[**Creating the Basic Routes**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#creating-the-basic-routes)

We will now create the routes to handle **getting all the bears** and **creating a bear**. This will both be handled using the /api/bears route. We’ll look at creating a bear first so that we have bears to work with.

**Creating a Bear**POST /api/bears

We will add the new route to handle POST and then test it using Postman.

// server.js

...

// ROUTES FOR OUR API

// =============================================================================

... // <-- route middleware and first route are here

// more routes for our API will happen here

// on routes that end in /bears

// ----------------------------------------------------

router.route('/bears')

// create a bear (accessed at POST <http://localhost:8080/api/bears>)

.post(function(req, res) {

var bear = new Bear(); // create a new instance of the Bear model

bear.name = req.body.name; // set the bears name (comes from the request)

// save the bear and check for errors

bear.save(function(err) {

if (err)

res.send(err);

res.json({ message: 'Bear created!' });

});

});

// REGISTER OUR ROUTES -------------------------------

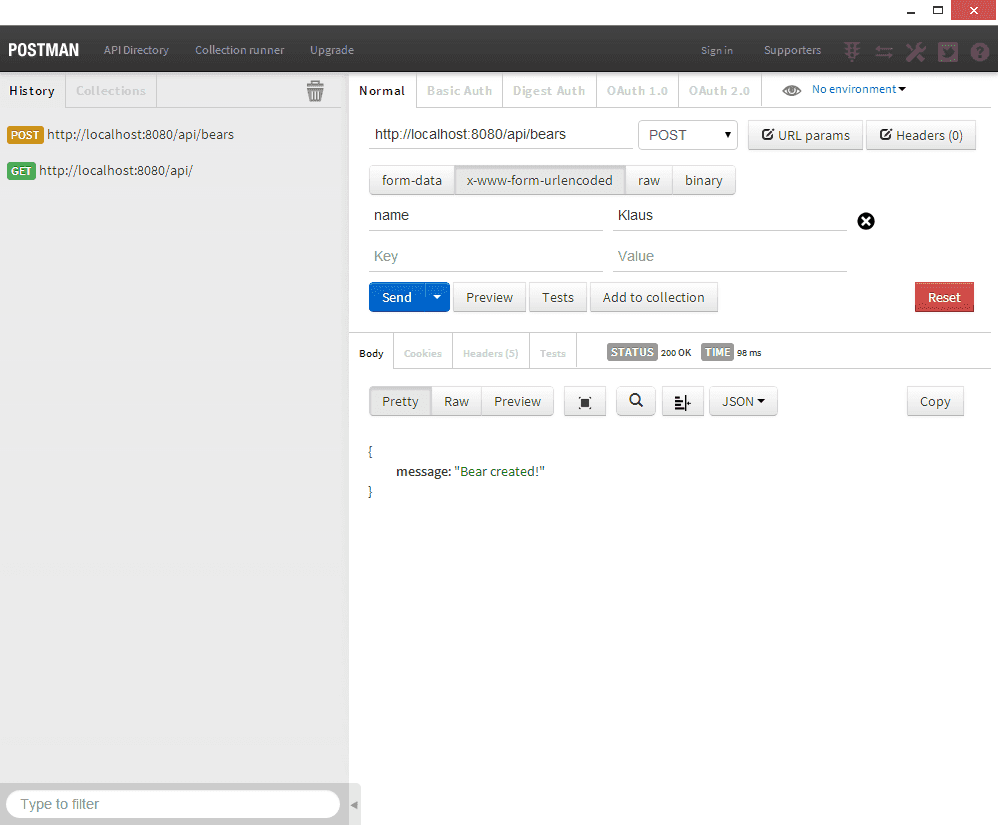
// all of our routes will be prefixed with /api

app.use('/api', router);

...

Now we have created the POST route for our application. We will use Express’s router.route() to handle multiple routes for the same URI. We are able to handle all the requests that end in /bears.

Let’s look at Postman now to create our bear.

[](https://cask.scotch.io/2014/04/node-api-postman-post-create-bear.png)

Notice that we are sending the name data as x-www-form-urlencoded. This will send all of our data to the Node server as query strings.

We get back a successful message that our bear has been created. Let’s handle the API route to get all the bears so that we can see the bear that just came into existence.

**Getting All Bears**GET /api/bears

This will be a simple route that we will add onto therouter.route() we created for the POST. With router.route(), we are able to chain together the different routes. This keeps our application clean and organized.

// server.js

...

// ROUTES FOR OUR API

// =============================================================================

... // <-- route middleware and first route are here

// more routes for our API will happen here

// on routes that end in /bears

// ----------------------------------------------------

router.route('/bears')

// create a bear (accessed at POST <http://localhost:8080/api/bears>)

.post(function(req, res) {

...

})

// get all the bears (accessed at GET <http://localhost:8080/api/bears>)

.get(function(req, res) {

Bear.find(function(err, bears) {

if (err)

res.send(err);

res.json(bears);

});

});

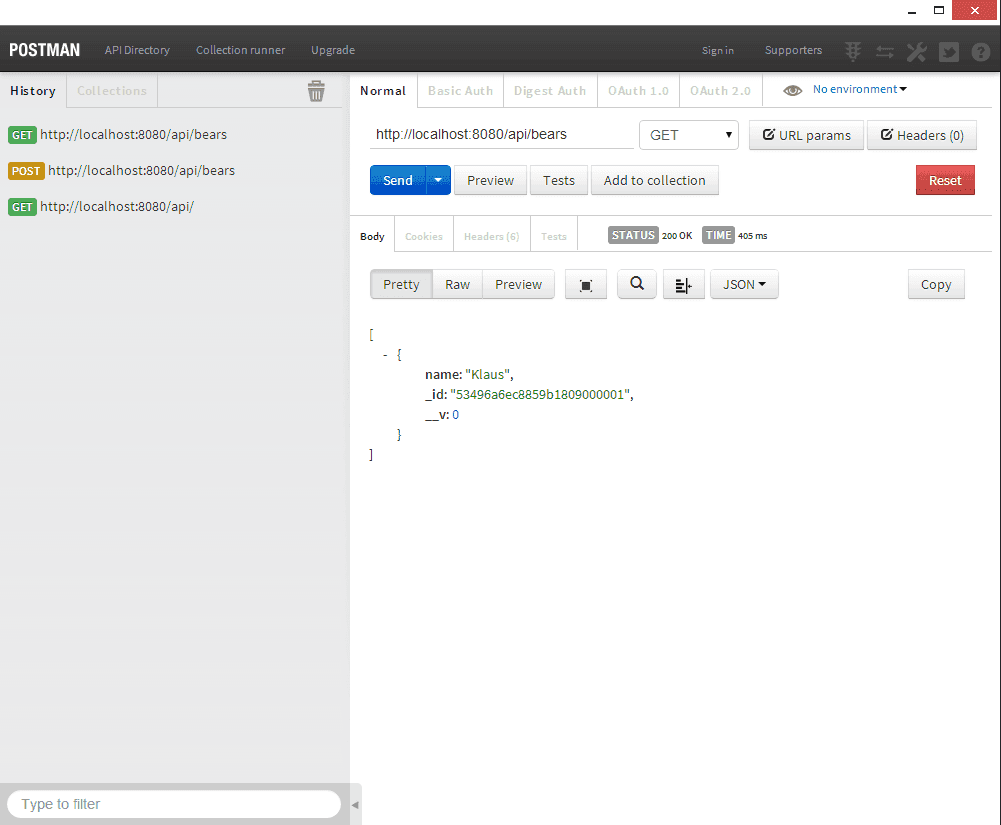
// REGISTER OUR ROUTES -------------------------------

// all of our routes will be prefixed with /api

app.use('/api', router);

...

Straightforward route. Just send a GET request tohttp://localhost:8080/api/bears and we’ll get all the bears back in JSON format.

[](https://cask.scotch.io/2014/04/node-api-postman-get-all.png)

[**Creating Routes for A Single Item**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#creating-routes-for-a-single-item)

We’ve handled the group for routes ending in /bears. Let’s now handle the routes for when we pass in a parameter like a bear’s id.

The things we’ll want to do for this route, which will end in /bears/:bear\_id will be:

* Get a single bear.
* Update a bear’s info.
* Delete a bear.

The :bear\_id from the request will be accessed thanks to that body-parser package we called earlier.

**Getting a Single Bear**GET /api/bears/:bear\_id

We’ll add another router.route() to handle all requests that have a :bear\_id attached to them.

// server.js

...

// ROUTES FOR OUR API

// =============================================================================

...

// on routes that end in /bears

// ----------------------------------------------------

router.route('/bears')

...

// on routes that end in /bears/:bear\_id

// ----------------------------------------------------

router.route('/bears/:bear\_id')

// get the bear with that id (accessed at GET <http://localhost:8080/api/bears/:bear_id>)

.get(function(req, res) {

Bear.findById(req.params.bear\_id, function(err, bear) {

if (err)

res.send(err);

res.json(bear);

});

});

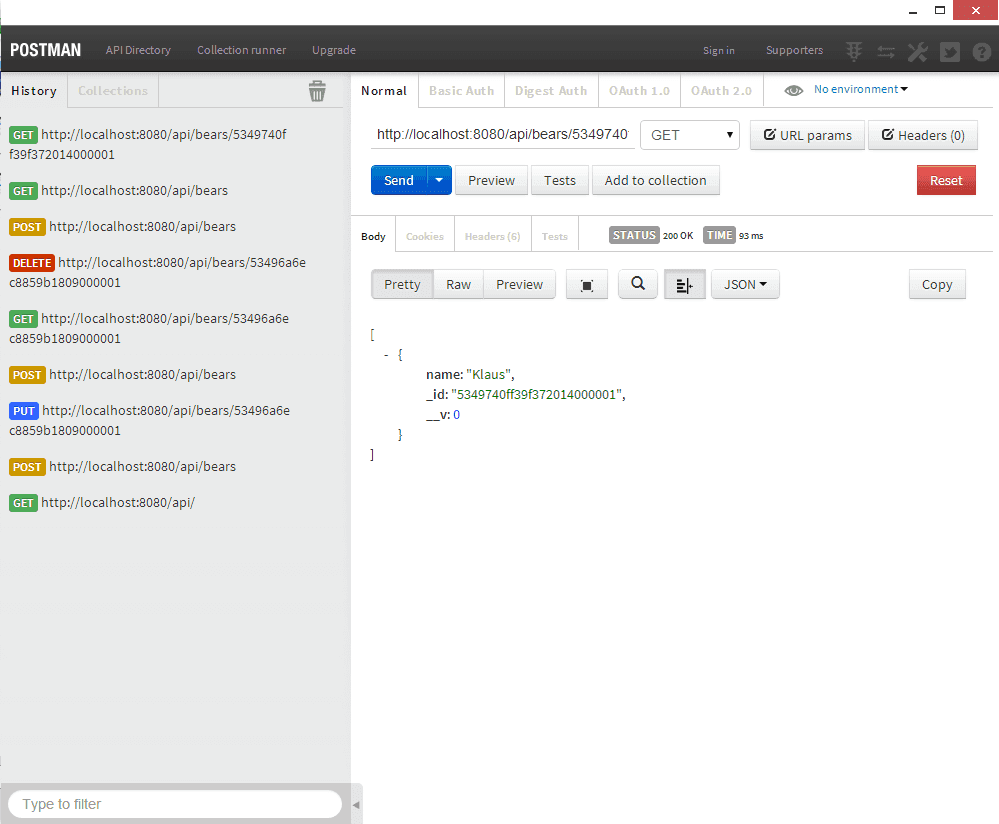
// REGISTER OUR ROUTES -------------------------------

// all of our routes will be prefixed with /api

app.use('/api', router);

...

From our call to get all the bears, we can see the long id of one of our bears. Let’s grab that id and test getting that single bear in Postman.

[](https://cask.scotch.io/2014/04/node-api-postman-get-single.png)

We can grab one bear from our API now! Let’s look at updating that bear’s name. Let’s say he wants to be more sophisiticated so we’ll rename him from Klaus to**Sir Klaus**.

**Updating a Bear’s Info**PUT /api/bears/:bear\_id

Let’s chain a route onto our this router.route() and add a .put().

// server.js

...

// on routes that end in /bears

// ----------------------------------------------------

router.route('/bears')

...

// on routes that end in /bears/:bear\_id

// ----------------------------------------------------

router.route('/bears/:bear\_id')

// get the bear with that id (accessed at GET <http://localhost:8080/api/bears/:bear_id>)

.get(function(req, res) {

...

})

// update the bear with this id (accessed at PUT <http://localhost:8080/api/bears/:bear_id>)

.put(function(req, res) {

// use our bear model to find the bear we want

Bear.findById(req.params.bear\_id, function(err, bear) {

if (err)

res.send(err);

bear.name = req.body.name; // update the bears info

// save the bear

bear.save(function(err) {

if (err)

res.send(err);

res.json({ message: 'Bear updated!' });

});

});

});

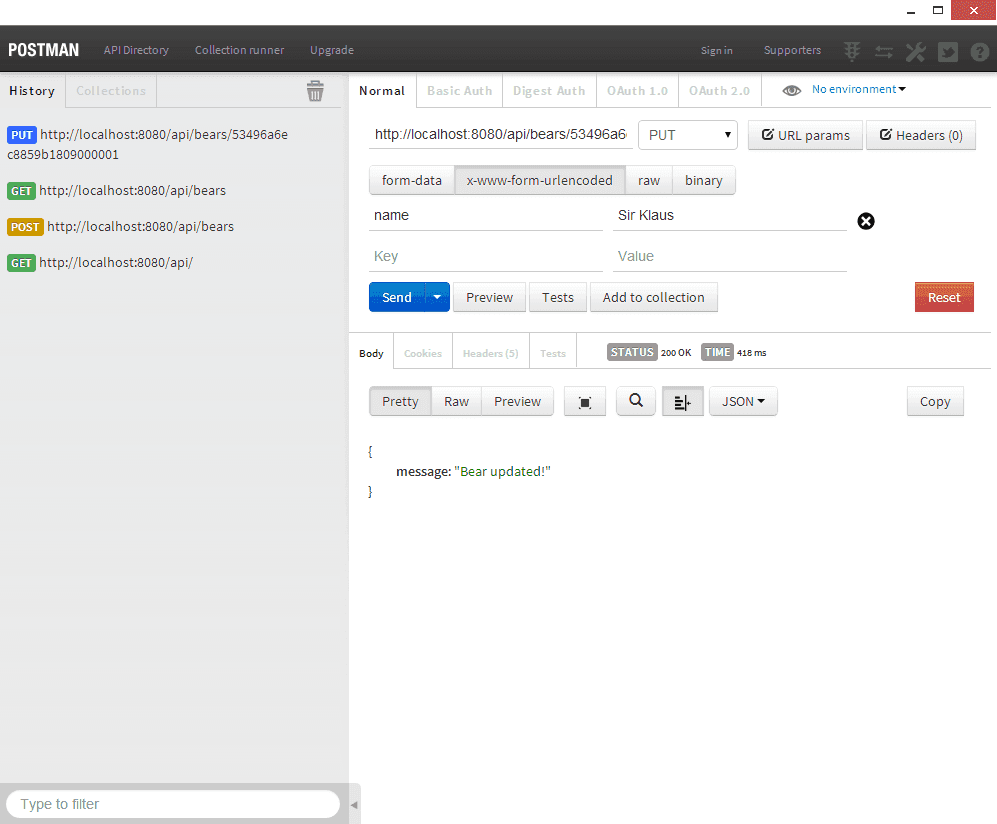
// REGISTER OUR ROUTES -------------------------------

// all of our routes will be prefixed with /api

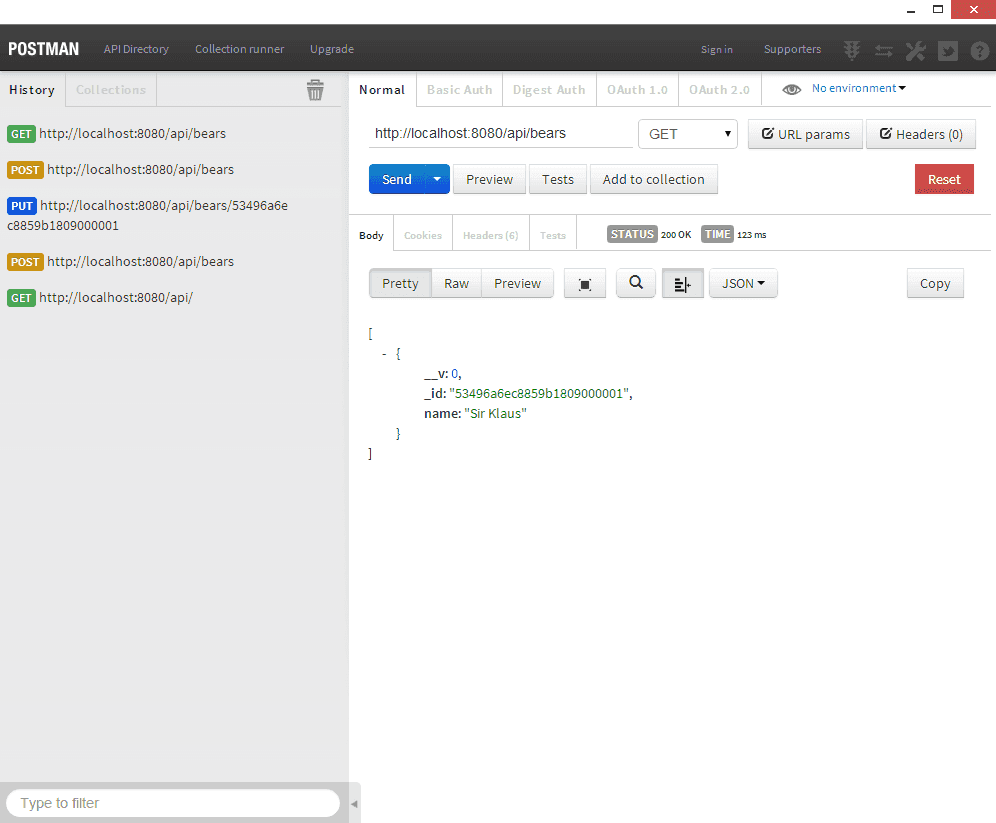
app.use('/api', router);

...

We will use the given id from the PUT request, grab that bear, make changes, and save him back to the database.

[](https://cask.scotch.io/2014/04/node-api-post-man-update-record.png)

We can also use the GET /api/bears call we used earlier to see that his name has changed.

[](https://cask.scotch.io/2014/04/node-api-postman-all-updated.png)

**Deleting a Bear**DELETE /api/bears/:bear\_id

When someone requests that a bear is deleted, all they have to do is send a DELETE to /api/bears/:bear\_id

Let’s add the code for deleting bears.

// server.js

...

// on routes that end in /bears

// ----------------------------------------------------

router.route('/bears')

...

// on routes that end in /bears/:bear\_id

// ----------------------------------------------------

router.route('/bears/:bear\_id')

// get the bear with that id (accessed at GET <http://localhost:8080/api/bears/:bear_id>)

.get(function(req, res) {

...

})

// update the bear with this id (accessed at PUT <http://localhost:8080/api/bears/:bear_id>)

.put(function(req, res) {

...

})

// delete the bear with this id (accessed at DELETE <http://localhost:8080/api/bears/:bear_id>)

.delete(function(req, res) {

Bear.remove({

\_id: req.params.bear\_id

}, function(err, bear) {

if (err)

res.send(err);

res.json({ message: 'Successfully deleted' });

});

});

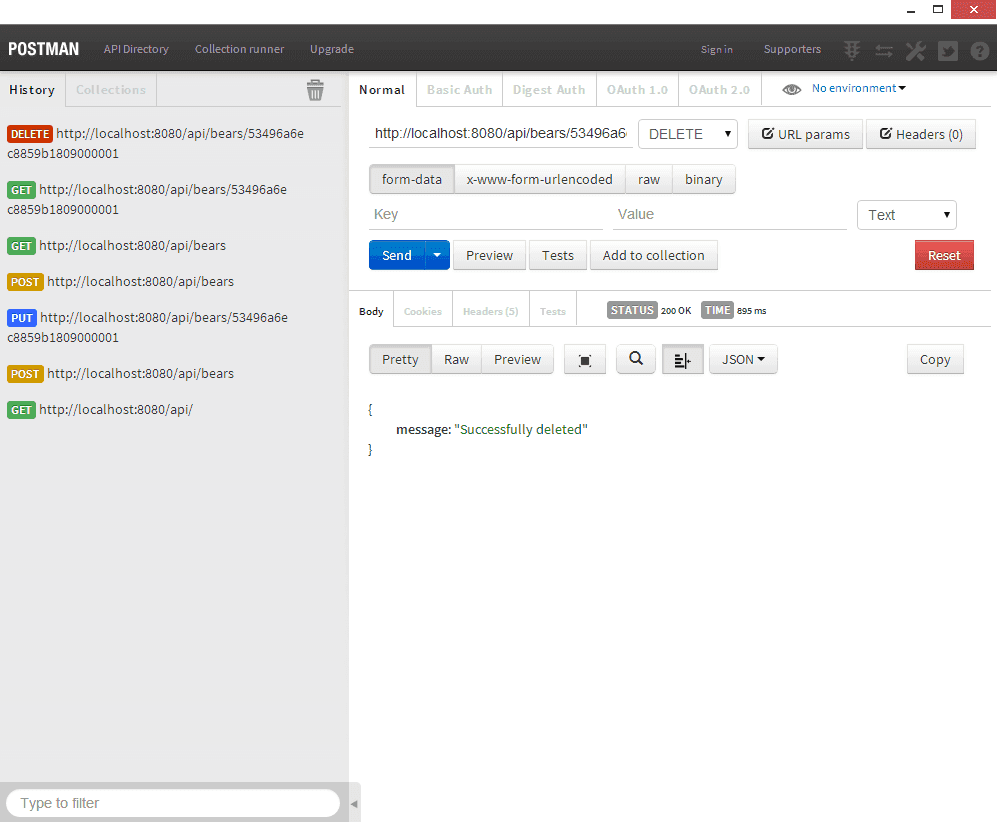
// REGISTER OUR ROUTES -------------------------------

// all of our routes will be prefixed with /api

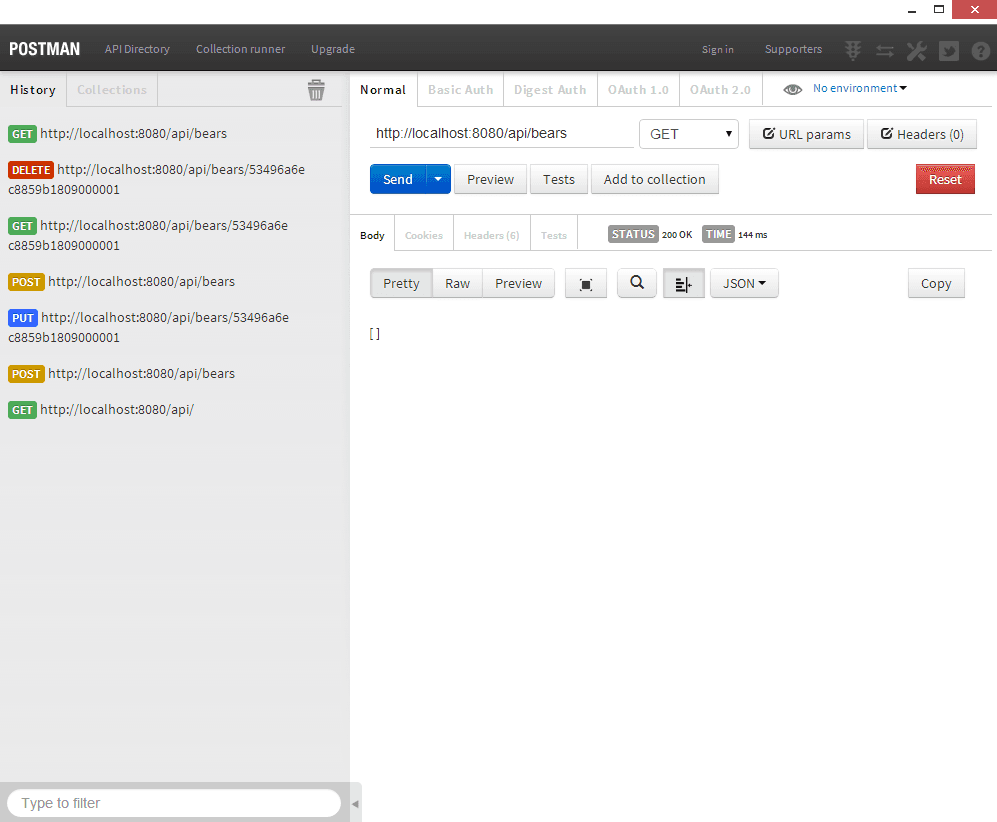
app.use('/api', router);

...

Now when we send a request to our API using DELETEwith the proper bear\_id, we’ll delete our bear from existence.

[](https://cask.scotch.io/2014/04/node-api-postman-delete.png)

When we try to get all the bears, there will be nothing left of them.

[](https://cask.scotch.io/2014/04/node-api-postman-get-all-nothing.png)

[**Conclusion**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#conclusion)

We now have the means to handle CRUD on a specific resource (our beloved bears) through our own API. Using the techniques above should be a good foundation to move into building larger and more robust APIs.

This has been a quick look at creating a Node API using Express 4. There are many more things you can do with your own APIs. You can add authentication, create better error messages, add different sections so you’re not just working with bears.

Sound off in the comments if you have any questions or would like to see any specific topics in the future.

[**Want More**](https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4#want-moremean?setting-up-a-mean-stack-single-page-applicationbuild-a-restful-api-using-node-and-express-4using-gruntjs-in-a-mean-stack-applicationauthenticate-a-node-api-with-tokens)