

Node is a powerful tool to get **JavaScript** on the server. Use Node to build a great website.

Get Started

(<https://bit.ly/2lkYUxR>)

 Code (<https://github.com/scotch-io/node-api>)

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 (</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/fhbjgbiflinjbdgghehdcddcbncdddomop>) in Chrome.

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

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> and http://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

(<https://hrd.cm/2k90ljr>)

HIRED

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 I want to get hired.



(<https://github.com/scotch-io>)



(<https://feeds.feedblitz.com/scotch-io>)

Monthly Best

Best Courses

your application (users, superheroes, beers, etc).

Make sure you have Node (<https://nodejs.org>) installed and let's get to it!

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.**

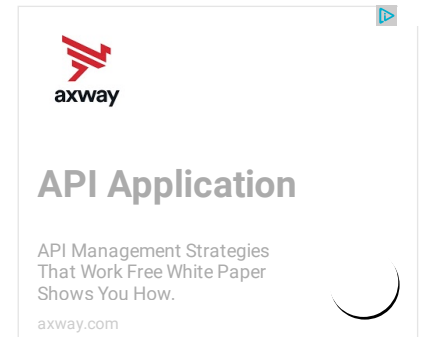
Here is our file structure. We won't need many files and we'll keep this very simple for demonstration purposes. When moving to a production or larger application, you'll want to separate things out into a good structure (like having your routes in their own file).

JAVASCRIPT

```
- app/
----- models/
----- bear.js // our bear model
- node_modules/ // created by npm. holds our dependencies/package
- 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.



(<https://hrd.cm/2k90ljr>)
HIRED

Companies find phenomenal people on Hired

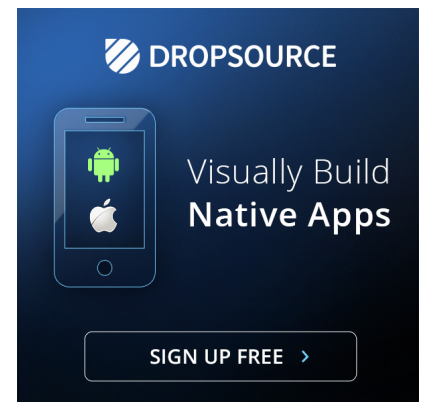
► Get Hired.

Subscribe & get our FREE eBook
Build Your First Node.js App



Email Address

Free Node Book



Build your data-driven and truly native iOS or Android app in days.
Use this drag and drop mobile app dev platform that easily integrates with any REST API.

(<https://synd.co/2mc7u3j>)

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// 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:

JAVASCRIPT

```
$ npm install
```

npm will now pull in all the packages defined into a `node_modules` folder in our project.

(<https://hrd.cm/2k90ljr>)

HIRED `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.

Better Matching for the Right Fit

I want to get hired.

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 bare (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.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// 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/)
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);
```

(<https://hrd.cm/2k90ljr>)

HIRED

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

Better Matching
for the Right Fit

I want to get hired.

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

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:

JAVASCRIPT

```
$ node server.js
```

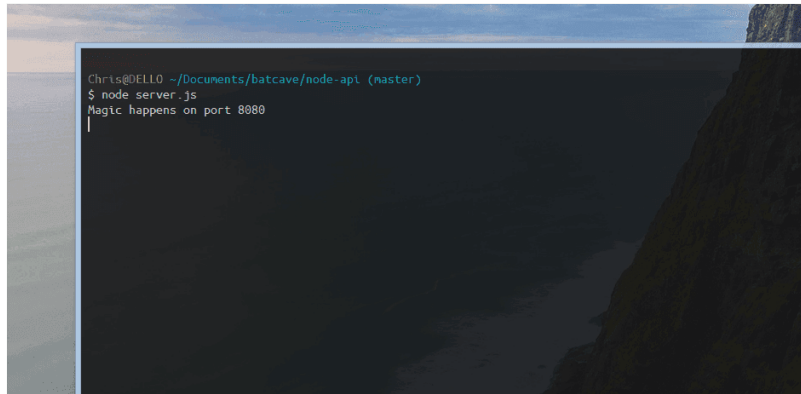
You should see your Node app start up and Express will create a server.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.



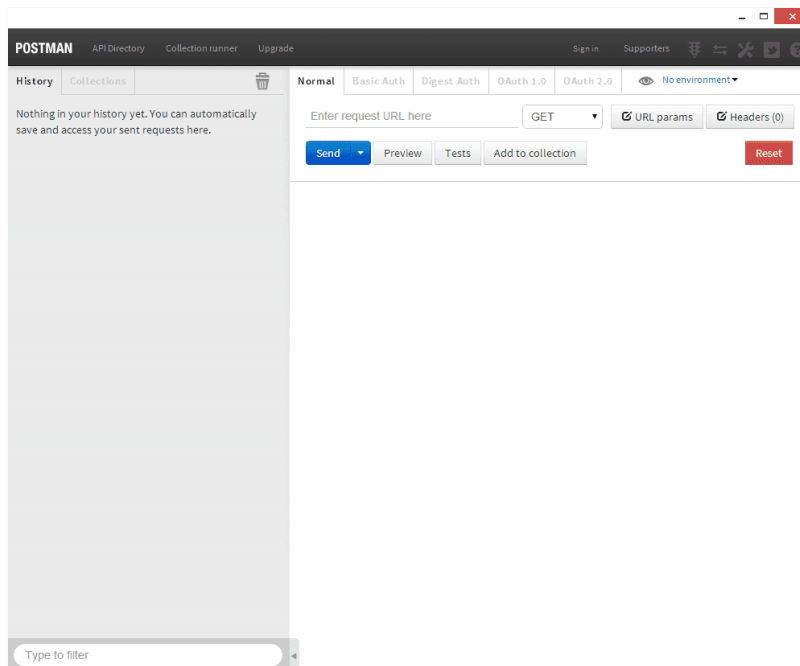
(<https://cask.scotch.io/2014/04/node-api-start-server.png>)

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://hrd.cm/2k90ljr>)

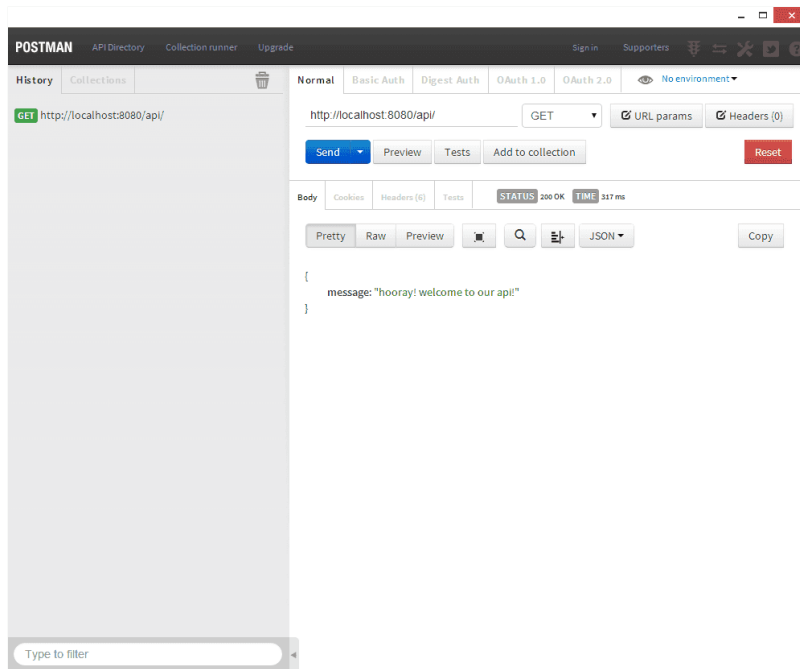
HIRED

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

Better Matching
for the Right Fit

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? Enter `http://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

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.

(<https://hrd.cm/2k90ljr>)

HIRED

CREATING OUR DATABASE AND CONNECTING

Better Matching
for the Right Fit

I want to get hired.

We will be using a database provided by Modulus (<https://modulus.io>). You can definitely create your own database and use it locally or use the awesome Mongolab (<https://mongolab.com/>). 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. In `server.js` in the Base Setup section, let's add these two lines.

JAVASCRIPT

```
// server.js

// BASE SETUP
// =====

...

var mongoose = require('mongoose');
mongoose.connect('mongodb://node:node@novus.modulismongo.net:27017/Iganic');

...
```

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.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// 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.

JAVASCRIPT

```
// 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!

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

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

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

(https://hrd.cm/2k90ljr)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// server.js

...

// ROUTES FOR OUR API
// =====

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

// 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://
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.

(<https://hrd.cm/2k90ljr>)

HIRED 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.

Better Matching
for the Right Fit

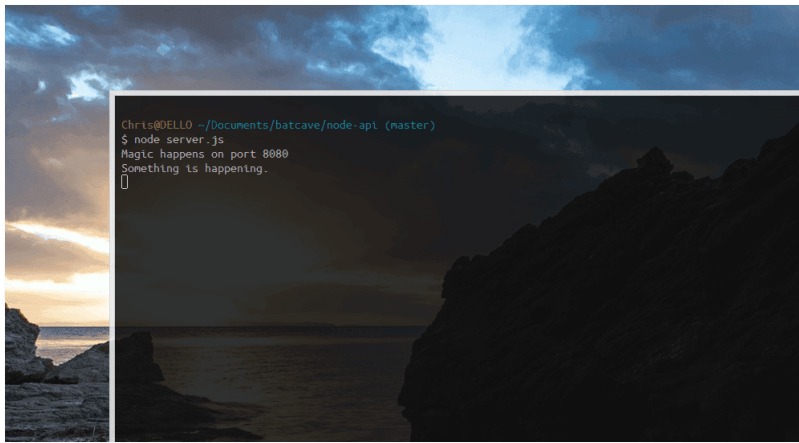
I want to get hired.

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.

(<https://hrd.cm/2k90ljr>)

HIRED

Creating the Basic Routes
Better Matching
for the Right Fit

I want to get hired.

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.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

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

        // save the bear and check for errors
        bear.save(function(err) {
            if (err)
                res.send(err);

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

    });

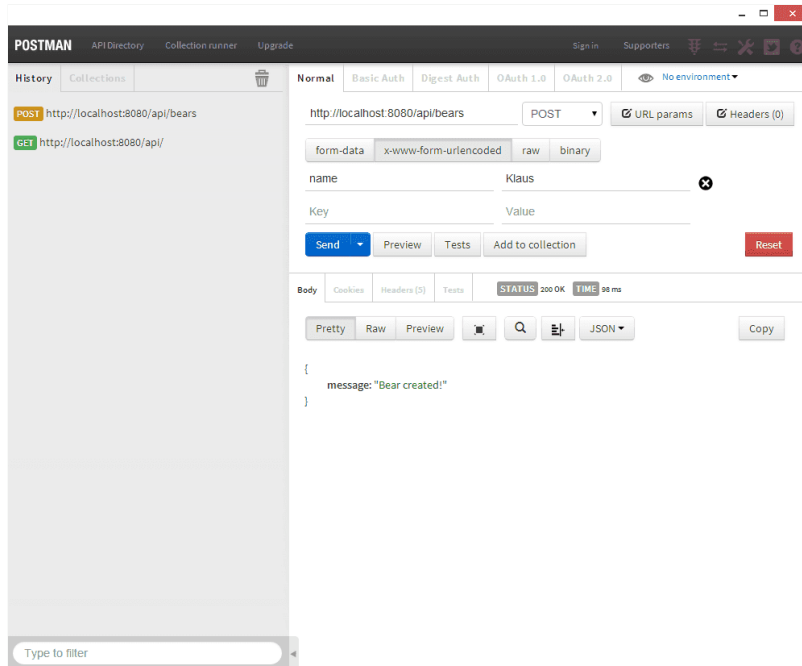
// REGISTER OUR ROUTES -----
// all of our routes will be prefixed with /api
app.use('/api', router);

...
```

(<https://hrd.cm/2k90ljr>)

HIRED
 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
**Better Matching
 for the Right Fit**
`/bears`.
I want to get hired.

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 the `router.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.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// 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);

...
```

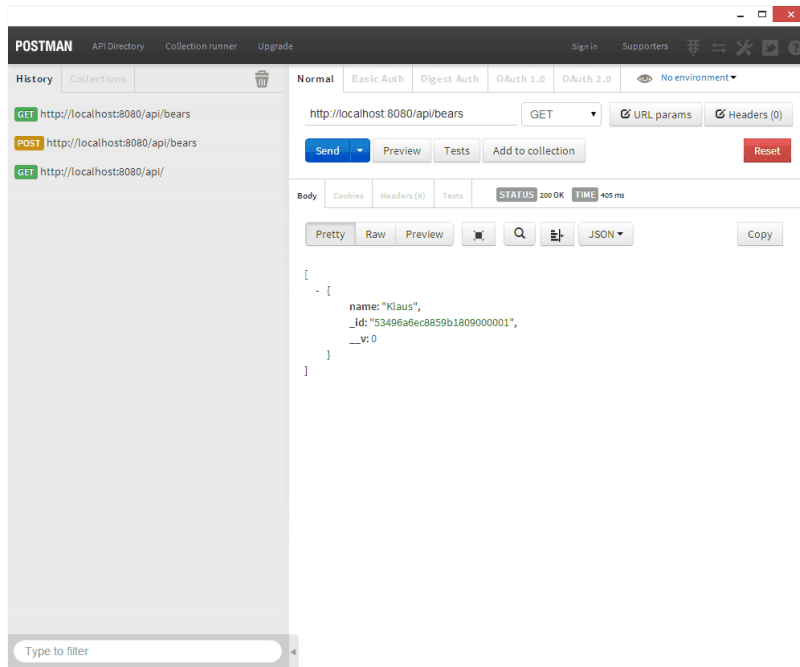
(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

Straightforward route. Just send a [GET](#) request to
<http://localhost:8080/api/bears> and we'll get all the bears back in
JSON format.

I want to get hired.



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

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.

(<https://hrd.cm/2k90ljr>)

GETTING A SINGLE BEAR `GET /API/BEARS/:BEAR_ID`

Better Matching for the Right Fit
I want to get hired.
We'll add another `router.route()` to handle all requests that have a `:bear_id` attached to them.

JAVASCRIPT

```
// 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/
  .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);

...
```

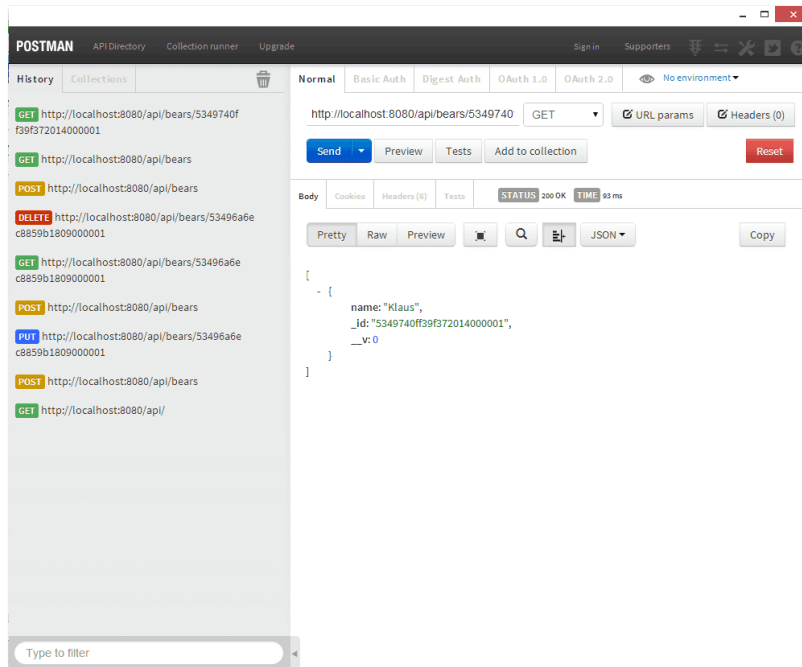
From our call to get all the bears, we can see the long id of one of our bears. Let's grab that id and test getting that single bear in

Postman.
(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.



(<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 sophisticated 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()`.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// 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/bears/:bear_id)
  .get(function(req, res) {
    ...
  })

  // update the bear with this id (accessed at PUT http://localhost:8080/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!' });
      });
    });
  });
});
```

(<https://hrd.cm/2k90ljr>)

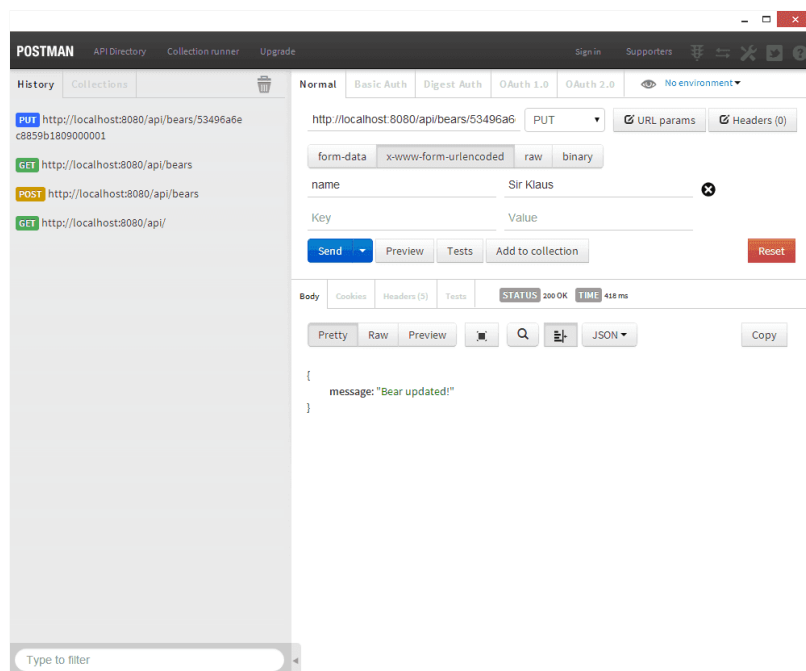
```
HIRED // REGISTER OUR ROUTES -----
// all of our routes will be prefixed with /api
app.use('/api', router);
```

Better Matching
for the Right Fit

I want to get hired.

...

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>)

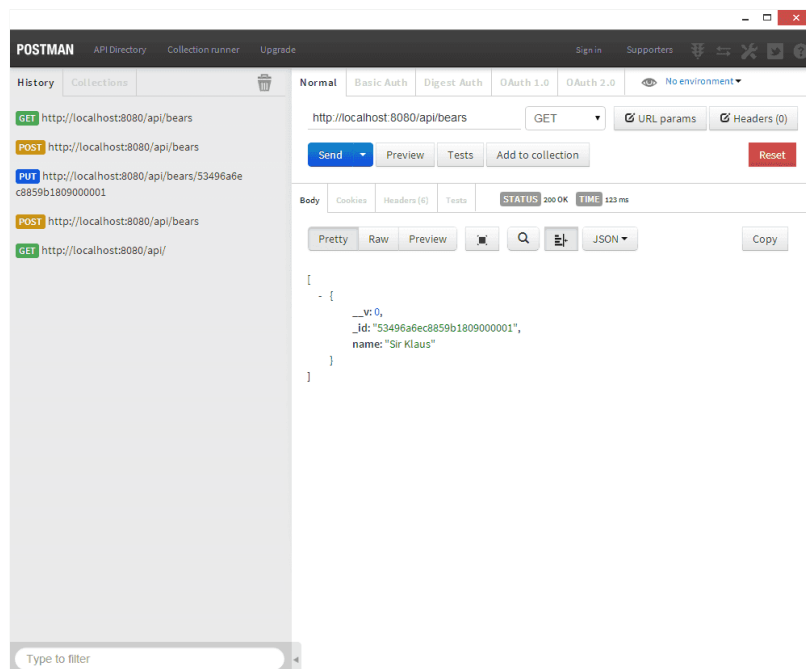
We can also use the `GET /api/bears` call we used earlier to see that his name has changed.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.



(<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.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

JAVASCRIPT

```
// 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);
```

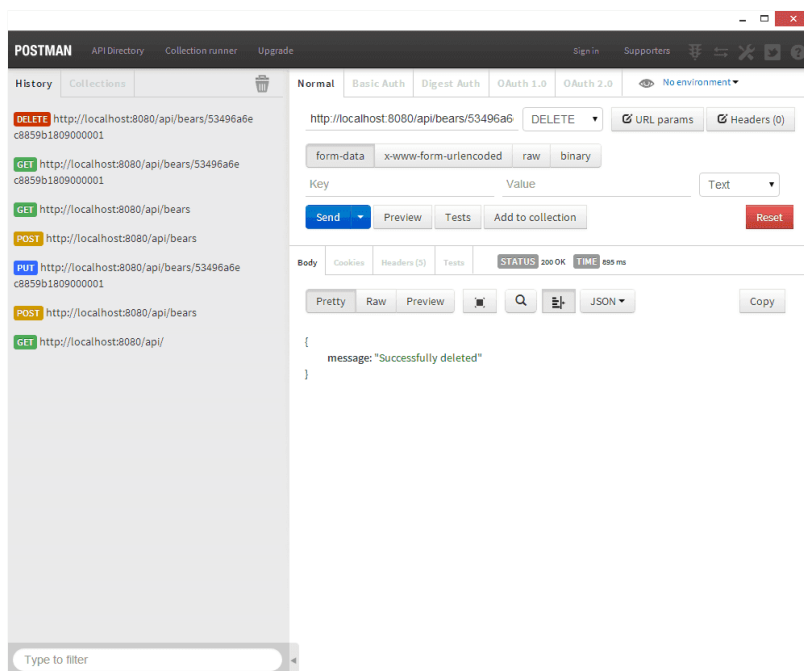
(https://hrd:cm/2k90ljr)

HIRED

Better Matching
for the Right Fit

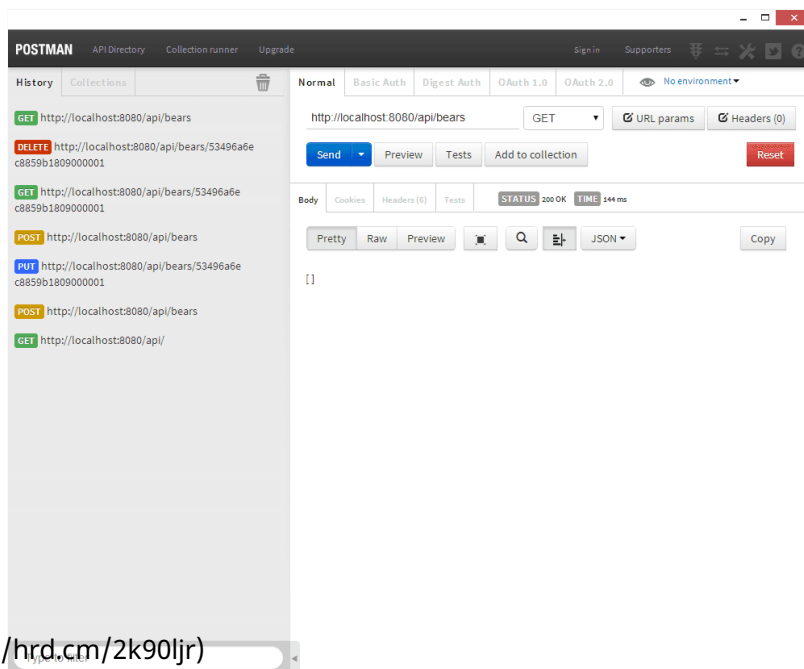
I want to get hired.

Now when we send a request to our API using `DELETE` with 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://hrd.cm/2k90ljr>)

Better Matching
for the Right Fit

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

I want to get hired.

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 MEAN? Setting Up a MEAN Stack Single Page Application Build a RESTful API Using Node and Express 4 Using GruntJS in a MEAN Stack Application Authenticate a Node API with Tokens



Code

API (<https://scotch.io/tag/api>) expressJS (<https://scotch.io/tag/expressjs>)

(<https://github.com/scotch-mean>)
MEAN (<https://scotch.io/tag/mean>) mongoDB (<https://scotch.io/tag/mongodb>)

(<https://hrd.cm/2k90ljr>)
node.js (<https://scotch.io/tag/node-js>) REST (<https://scotch.io/tag/rest>)

Better Matching
for the Right Fit

I want to get hired.

(https://pub.scotch.io/@chris)

io/node-

api)

Chris Sevilleja (https://pub.scotch.io/@chris)

Co-founder of Scotch.io: Slapping the keyboard until something good happens.
(https://plus.google.com/+ChrisSevilleja-scotch) (https://github.com/sevilayha)
(https://facebook.com/sevilayha) (https://twitter.com/chrisoneida)
(https://plus.google.com/+ChrisSevilleja-scotch) (https://instagram.com/sevilayha)

Next Up

Off Canvas Menus with CSS3 Transitions and Transforms

With the release of

(https://scotch.io/tutorials/off-canvas-menus-with-css3-transitions-and-transforms) (https://scotch.io/tag/css) ui/ux (https://scotch.io/tag/uiux)

Learn to Use the New Router in ExpressJS 4.0

With the release of

(https://scotch.io/tutorials/learn-to-use-the-new-router-in-expressjs-4) (https://scotch.io/tag/routing)

(https://hrd.cm/2k90ljr)

HIRED

Better Matching
for the Right Fit

I want to get hired.

261 Comments **Scotch** Login Recommend 24 Share

Sort by Best



Join the discussion...

**Iladarsda** • 3 years ago


An example of customer OAuth (user+pass+token) implementation (e.g. to serve multiple clients) would be very useful!

51 ^ | v • Reply • Share >

Connor James Leech  Iladarsda • 3 years ago

+1 I'm looking for a tutorial on how to secure a node.js API so my website and angular cordova app can both use it

6 ^ | v • Reply • Share >

**santi**  Iladarsda • 3 years ago

please!!

^ | v • Reply • Share >

Chris Sevilleja  **Bartender**  santi • 3 years ago

I'm currently working on one as we type. Just want to make sure that everything is solid. Definitely have to be careful we're not putting out bad information for security articles.


I wish I could give a timeline but I am working on it!

20 ^ | v • Reply • Share >

Teo  Chris Sevilleja • 3 years ago

Does this stuff is so complicated? I. not sure if i really want to continue learning this, if for an experienced developer in these technologies would take at least 2-3 months ...

6 ^ | v • Reply • Share >

Dardan  Chris Sevilleja • 2 years ago

Are you still working on this? Or is there a link you can share because I cant seem to find it and its been 5 months.

^ | v • Reply • Share >

Chris Sevilleja  **Bartender**  Dardan • 2 years ago

Yes. Sorry for the delay. I'm putting together this article along with a ton of other JavaScript, node, and angular articles.

Here's a good node package for doing token authentication.

They implement a good solution that is close to how the

tutorial I'm putting together will work:

<https://passwordless.net/>

Look out for that article next month. We're gonna do a couple weeks of JavaScript articles to coincide with our book release.

2 ^ | v • Reply • Share >

jwvanderbeck  Chris Sevilleja • 2 years ago

This still coming? REALLY would like to see how to setup a proper authenticated API.

1 ^ | v • Reply • Share >

Joshua Jung  Chris Sevilleja • 2 years ago

Hi Chris, do you have any update on the API authorization article or a recommended link to outside resources? Thanks!

^ | v • Reply • Share >

Chris Sevilleja  **Bartender**  Joshua Jung • 2 years ago

We have the concept ready to go in our book: MEAN Machine

<https://leanpub.com/mean-ma...>

We will be releasing the topic in article form some time in the

(https://hrd.cm/2k90ljr)

HIRED

Better Matching
for the Right Fit

I want to get hired.

next month though.

^ | v • Reply • Share ›

ClayM ➔ Chris Sevilleja • 2 years ago

How's this thing looking?

^ | v • Reply • Share ›



Matthijs ➔ Chris Sevilleja • 2 years ago

I'm currently building an ember.js app, so I would love to see this tutorial

^ | v • Reply • Share ›

Tim ➔ Chris Sevilleja • 3 years ago

I can't wait!!

^ | v • Reply • Share ›

This comment was deleted.

Chris Sevilleja Bartender ➔ Guest • 3 years ago

This is definitely on my to do list. It will be coming soon along with a lot more authentication stuff.

4 ^ | v • Reply • Share ›

Jesús Mur ➔ Chris Sevilleja • 3 years ago

It could be very useful for me :)

^ | v • Reply • Share ›

Jimmy Engström • 3 years ago

Very good tutorial!

I would love to see this using a TDD approach.

11 ^ | v • Reply • Share ›

Etienne Soulard Geoffrion • 3 years ago

is there a particular reason that when i run your project locally, i do not get any response from the server.. not even localhost:8080/api/

10 ^ | v • Reply • Share ›

Llewellyn Lovell ➔ Etienne Soulard Geoffrion • 2 years ago

for future users experiencing this problem, make sure you have created a mongolab user and that your database user and password have been set correctly in the connecting url i.e. mongodb://<dbuser>:<dbpassword>@

1 ^ | v • Reply • Share ›

The Mother of Joseph Beuys ➔ Llewellyn Lovell • 2 years ago

A simpler alternative is just to create & run a db within your project itself. Here's how (I called my project folder "node_api"):

1. Open a CLI window and type the following

```
cd /blah/blah/node_api<enter>
mkdir data<enter>
mongod --dbpath /blah/blah/node_api<enter>
```
2. A bunch of text should appear, you will see "I NETWORK [initandlisten]" alot;
3. Open a new CLI window and type: mongo<enter>
4. Hopefully you will see "MongoDB shell version: 3.0.1 connecting to: test" If not, go to the root folder of your OS and make a directory called "data" and then repeat from step 3. above
5. In the second "mongo<enter>" CLI window type: use node_api*
6. You should see "switched to db node_api"
7. Open server.js and change the "mongoose.connect..." line to the following:

```
mongoose.connect('mongodb://localhost/node_api');*
```
8. In a third [new?] CLI window ctrl+C (if necessary), cd to blah/blah/node_api* (if necessary) and then run "node server.js" again.

You should now have a mongo db located in node_api/data and be

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

connecting to it via server.js

* substitute "node_api" for whatever the name of your project folder is.

3 ^ | v • Reply • Share ›



Vayodya → The Mother of Joseph Beuys • a month ago

Thanks.. it's now Working fine

^ | v • Reply • Share ›

syd → The Mother of Joseph Beuys • a year ago

Just wanted to check, why did you need to create the folder "data". When i did the mongod -dbpath /blah/blah command, it installed the files in the main folder and not the data folder. is this the intended result?

^ | v • Reply • Share ›

syd → The Mother of Joseph Beuys • a year ago

I just signed up an account to say two things.

It worked! and

Thank you

The tutorial skipped out the part on connecting to mongodb. And mongodb tutorial out there teaches you how to install it on the root drive. Nobody teaches how to connect between the two... that is until I saw your comments.

^ | v • Reply • Share ›



Triana → The Mother of Joseph Beuys • a year ago

Thank you this got me on the right track!

to make it work I had to use:

```
mongoose.connect('mongodb://localhost:27017/node_api');
```

^ | v • Reply • Share ›

markeban → The Mother of Joseph Beuys • a year ago

Thank you!!!!!! So incredibly helpful.

^ | v • Reply • Share ›



Justin → Llewellyn Lovell • 2 years ago

I am not connecting to localhost either and it is because there is a problem with mongoose. I created a database using Modulus like Chris said and added the URI in mongoose.connect but it is not connecting. I also tried copy/pasting his own mongoose.connect code and it is not working. any ideas? Am I supposed to be running Mongod on another command line prompt? Or am I supposed to connect to a database just with those two lines of code?

^ | v • Reply • Share ›

Chris Sevilleja Bartender → Justin • 2 years ago

Did you make sure you changed the <user> and <pass> in the database URI for modulus? The URI they provide you doesn't have that info in it. And the URI I provided won't work since I revoked that access. A lot of people used it for saving random info for their personal projects sadly.

If you work locally, then yes, you need to have mongod running.

^ | v • Reply • Share ›

Soham Chakraborty → Chris Sevilleja • a year ago

I am having the same problem Chris - i entered the mongolab URI correctly - yet , its not working- when I go to test in Postman, its showing - loading and in console its showing - GET /api/users - - ms - -

^ | v • Reply • Share ›

kalexander → Soham Chakraborty • a year ago

This can be solved by updating mongoose.

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.

1. Go to package.json and change mongoose version to * (star)
 2. Run "npm install" in project folder
 3. ????
 4. Profit!
- 2 ^ | v • Reply • Share ›

Kevyn Hale ➔ kallexander • 3 months ago

You saved, fixed all my issues.

^ | v • Reply • Share ›

A-CURSED-GINGER ➔ kallexander • 5 months ago

This worked perfectly!

^ | v • Reply • Share ›

John Schulz ➔ kallexander • a year ago

Thank you, this worked for me!

^ | v • Reply • Share ›

Atilio • 3 years ago

Hi, great tutorial, for some reason I can't save the name of my bear, whenever I POST a new bear i get a successful json response but when I retrieve all bears all I get are objects with _id and _v and no name field, do you have any idea why this would be happening? I can add a "name" field successfully from Robomongo and the mongo shell and don't know what could be happening

12 ^ | v • Reply • Share ›

Polished85 ➔ Atilio • 2 years ago

Had the same problem. Make sure to click 'x-www-form-urlencoded' before sending a POST or PUT request.

2 ^ | v • Reply • Share ›

Marlon Yamil Uclés Hernández ➔ Polished85 • a year ago

Have the same problem too...

^ | v • Reply • Share ›

Glass Steagall ➔ Polished85 • 2 years ago

How do you set this in code?

^ | v • Reply • Share ›

Polished85 ➔ Glass Steagall • 2 years ago

With jQuery, use \$post to send as 'x-www-form-encoded' and \$ajax to send as json/xml

^ | v • Reply • Share ›

Charlie • 3 years ago

Great Article! How can I enable CrossDomain in this new router?

6 ^ | v • Reply • Share ›

abr4xas ➔ Charlie • 2 years ago

```
router.use(function(req, res, next) {
  console.log('Something AWESOME is happening...');
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "X-Requested-With");
  res.header('Access-Control-Allow-Headers', 'Content-Type');
  next(); // make sure we go to the next routes and don't stop here
});
```

2 ^ | v • Reply • Share ›

(<https://hrd.cm/2k90ljr>)

HIRED

Hardik Shah ➔ abr4xas • a year ago

That's a great shortcut!

^ | v • Reply • Share ›

Better Matching
for the Right Fit

Michael Battle ➔ Charlie • 3 years ago

Hi Charlie - use this: <https://www.npmjs.org/packa...>

I want to get hired. 2 ^ | v • Reply • Share ›

Hardik Shah → Michael Battle • a year ago

That's a great solution. I added "cors": "2.7.1" under dependencies under my package.json for Node.js API and followed the instruction over the link and it works fine.

Thanks Mike!

1 ^ | v • Reply • Share >



Guest • 2 years ago

Great tutorial! I'm having an issue with the req.body when I post to /bears. I'm using a local mongodb and it's successfully submitting, but the req.body is empty. It's submitting json and I've included the bodyParser middleware. Any ideas?

5 ^ | v • Reply • Share >

Max Norton → Guest • 3 months ago

I had the same problem. I fixed it by removing express, mongoose and body-parser as dependencies in the package.json and then re-installing the latest versions of them (npm install -save express etc...). Using the old npm packages mentioned in the article won't work.

^ | v • Reply • Share >

Juan Pablo Badino → Max Norton • 25 days ago

Thanks bud.

^ | v • Reply • Share >

Griffin Wiebel → Max Norton • 2 months ago

This helped me. Wow, figuring out that problem was a pain in the butt.

^ | v • Reply • Share >

Travis Miller → Guest • 2 years ago

I'm having the same problem.

^ | v • Reply • Share >

Josefreak Salem → Travis Miller • 2 years ago

The tutorial has a mistake. If you're using request.body, you shouldn't use x-www-form-urlencoded, you should just send raw json. In postman, make sure you set key/value in the header to Content-Type, application/json, then click the 'raw' button, then add the json: example {"name": "Josefreak"}. This will work with both POST and PUT sections of the tutorial.

3 ^ | v • Reply • Share >

danconiaus → Josefreak Salem • 3 months ago

You, my friend, are a lifesaver! Thanks for taking the time to post your solution.

^ | v • Reply • Share >

Sponsored Links

Gamers around the world have been waiting for this game!

(Forge Of Empires - Free Online Game)

The best Strategy Game of 2017!

(Vikings: Free Online Game)

How this app teaches you a language in 3 weeks! (Babbel)

56 of the Worst Cars of All Time (Carophile)

(<https://hrd.cm/2k90ljr>)

HIRED

Better Matching
for the Right Fit

I want to get hired.