<LABS/>

Search

Create a REST API With Node.js

May 14th, 2014

By Brittney Kernan

@brittneykernan

For a pitch, we prototyped one of our concepts with a Natural Language search feature. I quickly implemented a REST style GET call on search submit with Node.js via major article skimming. I still didn't have a clear mental map of how Node.js worked with a REST API, so I broke it down into a presentation for the BSS Tech team. You can follow along, or skim.

PHP vs Node.js REST Components

If you've come from PHP, your brain may already hold a diagram of what a REST API looks like in that language. Let's make one for Node.js:

PHP

- Apache - HTTP server
- .php controllers - Controls what your app does with data
- Content-Type JSON Headers - Sends your data back readable
- .htaccess - Map URLs to controller action methods for pretty URLs
- MySQLi - Interfaces with Data
- MySQL - Stores data

Node.js

- Node.js - HTTP Server
- JavaScript modules - Controllers for what your app does with data
- Express - Framework to easily build for web with Node.js
- Routes - Map URLs to actions for pretty URLs
- Mongoose.js - Interfaces with data
- Mongo - Stores data

Start a Node Server

Install Node first. Make a directory to play in, ex: *noderest*, and put *server.js* in the root.

server.js

```
1 var http = require('http');
2 http.createServer(function (req, res) {
3   res.writeHead(200, {'Content-Type': 'text/plain'});
4   res.end('Hello New York\n');
5 }).listen(3001);
6 console.log('Server running at http://localhost:3001/');
```

What's going on here

HTTP is the module we'll use to read the incoming Requests' HTTP action - GET, POST, PUT or DELETE - and Headers. It will also create a server for us, listening for Requests at the available port of your choice (here 3001). Pass `createServer()` a callback function that takes two arguments `req` and `res`. `req` for Request and `res` for Response. Add a `console.dir()` or two to print to Terminal what these two variables hold. Response is an object you can use to reply back to the Request. Forget to return

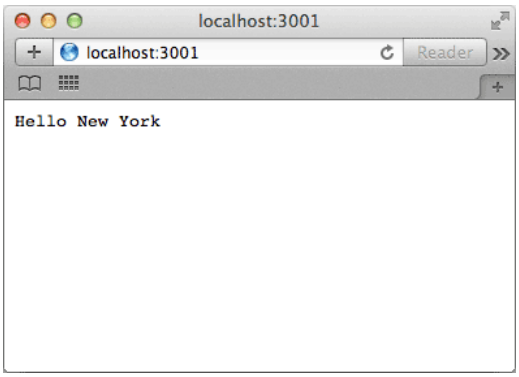
something with Response and your Request will simply time out. Here any URL path we type at http://localhost:3001/ will print back *Hello New York* with a *OK* header as plain text content.

Try it

In Terminal:

```
1 $ cd noderest/
2 $ node server.js
```

Terminal should print `Server running at http://127.0.0.1:3001/` back at you. So easy (as long as you didn't have anything else running in that port). Open that URL. You should see `Hello New York`.



Add Express

Why add Express? Same reason you use jQuery, makes things faster and easier. What the hell is it? I read the home page maybe 5 times and used it in 3 prototypes before my brain mapped it.

To add Express as a dependency to your project, we'll need to first install NPM (it should have installed with Node actually, if not). NPM will read our *package.json* file saved to the root of our project directory, and then download and install Express for us.

package.json

```
1 {
2   "name": "noderest",
3   "description": "Demo REST API with Node",
4   "version": "0.0.1",
5   "private": true,
6   "dependencies": {
7     "express": "3.x"
8   }
9 }
```

Install Express in Terminal

```
1 $ npm install
```

Watch NPM in Terminal fetch the files. Your directory now holds a new folder called *node_modules*, with the *express* folder inside. Let's use Express.

Use Express

Like the HTTP module you required in your first version of *server.js*, you'll require Express. Open *server.js*, clear it out. Add:

```
1 var express = require('express');
2 var app = express();
3 app.get('/', function(req, res) {
4   res.send('Hello Seattle\n');
5 });
6 app.listen(3001);
7 console.log('Listening on port 3001...');
```

What's changed

Nothing! Not really. Our server is actually doing the same thing, though. Listening to HTTP requests sent to locally to port 3001 and returning updated text *Hello Seattle*. Only difference is we now have Express' framework to work with - a slightly cleaner and faster way to handle requests and set up our server. The `app` variable holds an instance of the Express application object, which has many useful methods.

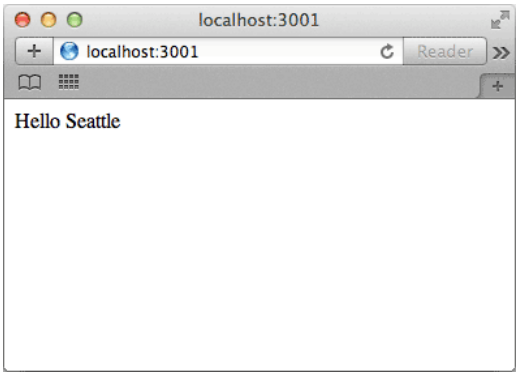
Run Express

Changes to Node.js server files require a full server restart in order to be run the updated version.

In Terminal, use `CTRL-C` to stop your running Node server. Restart with `node server.js`.

You'll see Terminal respond with *Listening on port 3001*. And *Hello Seattle* at your localhost in the browser. You may notice the font family is different than before.

Express is sending a text/html Content-Type Header instead of text/plain (Express is for web after all).



REST

We're ready to build our REST API. We'll need:

- A URL route schema to map requests to app actions
- A Controller to handle each action
- Data to respond with
- Place to store the data
- Interface to access and change data

Routes

Routes are the predefined URL paths your API responds to. Think of each Route as listening to three parts:

- A specific HTTP Action
- A specific URL path
- A handler method

```
1 app.get('/', function(req, res) {
2     res.send('Return JSON or HTML View');
3 });
```

This example of routing handles all GET Requests. The URL path is the root of the site, the homepage. The handling method is an anonymous function, and responds with plain text in this case.

Request

```
1 app.get('/musician/:name', function(req, res) {
2
3    // Get /musician/Matt
4    console.log(req.params.name)
5    // => Matt
6
7    res.send('{"id": 1,"name":"Matt",
8      "band":"BBQ Brawlers"}');
9 });
```

Paths are string patterns broken up by `/` and `:`, and translated into RegEx by Express. In this example, `:name` is anything that comes after `/musician/` in the url. This parameter will be available in our Request object `req` under the params property. It will be keyed the same name as our param in the route path.

FYI `req.body` can also be used to refer to the HTTP Request Body. You'll need to tell Express to look out for Requests Body's in your *server.js* file via `configure()`. We'll do this later.

```
1 app.configure(function(){
2     app.use(express.bodyParser());
3 });
```

Response

`res.send()` is awesome. You can pass this method JSON, Strings and HTML, and it will interoperate and send back the appropriate Content-Type and other Headers for you. If you want control, that's cool too. To send a custom HTTP Status pass that number in before the response content. Add other headers, use `res.set()` to add in your headers before calling `res.send()`.

```
1 res.send({ "some": "json" });
2 res.send("html for Maximum Pain's web page");
3 res.send(404, 'No musicians here');
4 res.send(500, { error: 'you blew it' });
5 res.send(200);
```

There's much more to Express than detailed here. For more read the well designed and written docs.

Routes.js

Add a new file at the root called *routes.js*. Drop in this routing module:

```
1 module.exports = function(app){
2     var musicians = require('./controllers/musicians');
3     app.get('/musicians', musicians.findAll);
4     app.get('/musicians/:id', musicians.findById);
5     app.post('/musicians', musicians.add);
6     app.put('/musicians/:id', musicians.update);
7     app.delete('/musicians/:id', musicians.delete);
8 }
```

Here we are working with a Musician data model. We've created a Musicians Controller and placed all our Request event handling methods inside the controller. The main REST HTTP actions are handled. We've modeled our URL routes off of REST API conventions, and named our handling methods clearly.

Controllers

To keep code organized, create a folder called *controllers* inside your project directory. Create a new file inside of that called *musicians.js*. We'll add each request handling method for Musicians data to this file one by one. For now add these placeholders to *musicians.js* so we can restart the server without errors:

```
1 exports.findAll = function() {};
2 exports.findById = function() {};
3 exports.add = function() {};
4 exports.update = function() {};
5 exports.delete = function() {};
```

Our Controller exports all of these methosd so the router can refer to them.

Find All

Update `findAll`'s definition' to the function below:

```
1 exports.findAll = function(req, res){
2   res.send([{
3     "id": 1,
4     "name": "Max",
5     "band": "Maximum Pain",
6     "instrument": "guitar"
7   }]);
8 };
```

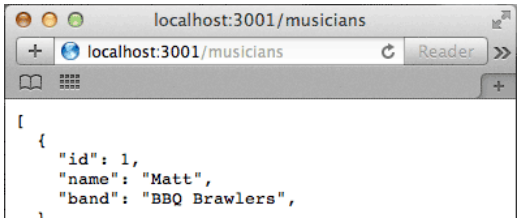
Like the anonymous functions we declared in our *server.js*, request handler methods accept `req` and `res` arguments and return JSON with `res.send()`. Our JSON is hard coded, which will do for now. Let's test this route in our app.

Import Routes and JS Modules

Update *server.js* to require our routes file. The `.js` file extension can be omitted.

```
1 var express = require('express');
2
3 var app = express();
4
5 require('./routes')(app);
6
7 app.listen(3001);
8 console.log("Jammin\' on port 3001...");
```

`CTRL+C` to stop the Node server and `node server.js` to restart. You should now see JSON at localhost:3001/musicians.



```
  ]
```

Mongo

Data would be more fun to play with. I made fake bands for all the techers on the team for my musicians data. Put thought into it.

Start Mongo

Mongo Database felt right for my light data and it is super fast to set up (if you already have Mongo installed, install it if not, yuck). Run `mongod` in another Terminal tab if it's not running already.

Mongoose.js

Mongo is easy enough to communicate with, but if you like a little bit more structure to your data and data interface, try a Mongo Driver. I used Mongoose.js because I liked the Schema style of defining document structure.

Ask NPM to install this dependency for you, and update your package.json file with this dependency for you with the `--save-dev` option.

```
1 npm install mongoose --save-dev
```

Use Mongoose.js

Update *server.js*:

```
1  var express = require('express'),
2  mongoose = require('mongoose'),
3  fs = require('fs');
4
5  var mongoUri = 'mongodb://localhost/noderest';
6  mongoose.connect(mongoUri);
7  var db = mongoose.connection;
8  db.on('error', function () {
9    throw new Error('unable to connect to database at ' + mongoUri);
10 });
11
12 var app = express();
13
14 app.configure(function(){
15   app.use(express.bodyParser());
16 });
17
18 require('./models/musician');
19 require('./routes')(app);
20
21 app.listen(3001);
22 console.log('Listening on port 3001...');
```

We're requiring the Mongoose module which will communicate with Mongo for us. The `mongoUri` is a location to the Mongo DB that Mongoose will create if there is not one there already. We added an error handler there to help debug issues connecting to Mongo collections. We also configured Express to parse requests' bodies (we'll use that for POST requests). Lastly, we require the Musician model which we'll make next.

Define Data Models

Create a new folder called *models* and add a new file *musician.js* for our Musician Model.

```
1  var mongoose = require('mongoose'),
2  Schema = mongoose.Schema;
3
4  var MusicianSchema = new Schema({
5    name: String,
6    band: String,
7    instrument: String
8  });
9
10 mongoose.model('Musician', MusicianSchema);
```

Require Mongoose into this file, and create a new Schema object. This schema helps Mongoose make sure it's getting and setting the right and well-formed data from and to the Mongo collection. Our schema has three String properties which define a Musician object. The last line creates the Musician model object, with built in Mongo interfacing methods. We'll refer to this Musician object in other files.

Find All

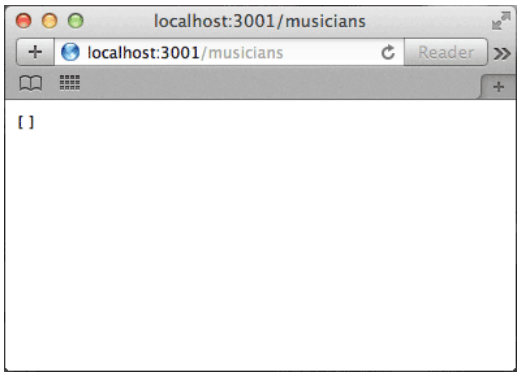
Update *controllers/musicians.js* to require Mongoose, so we can create an instance of our Musician model to work with. Update `findAll()` to query Mongo with the `find()` data model method.

```
1  var mongoose = require('mongoose'),
2  Musician = mongoose.model('Musician');
3
4  exports.findAll = function(req, res){
5    Musician.find({},function(err, results) {
6      return res.send(results);
7    });
8  };
9  exports.findById = function() {};
10 exports.add = function() {};
11 exports.update = function() {};
12 exports.delete = function() {};
```

Passing `find() {}` means we are not filtering data by any of its properties, so please return all of it. Once Mongoose looks up the data it returns an error message and a result set. Use `res.send()` to return the raw results.

Start Mongoose

Restart the server with `CTRL+C` and `node server.js`. Visit the API endpoint for all musicians localhost:3001/musicians. You'll get JSON data back, in the form of an empty array.



Data

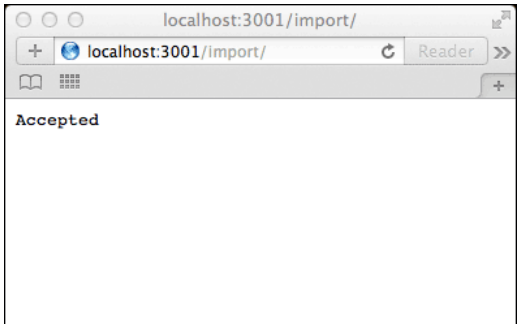
Since I didn't feel like messing with Mongo command-line, I decided to import musician data with our REST API. Add a new route endpoint to *routes.js*.

```
1 app.get('/import', musicians.import);
```

Now to define the import method in our Musicians Controller *controllers/musicians.js*:

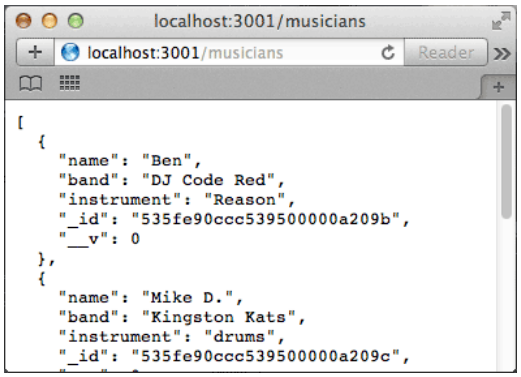
```
1  exports.import = function(req, res){
2    Musician.create(
3      { "name": "Ben", "band": "DJ Code Red", "instrument": "Reason" },
4      { "name": "Mike D.","band": "Kingston Kats", "instrument": "drums" },
5      { "name": "Eric", "band": "Eric", "instrument": "piano" },
6      { "name": "Paul", "band": "The Eyeliner", "instrument": "guitar" }
7    , function (err) {
8      if (err) return console.log(err);
9      return res.send(202);
10   });
11 };
```

This import method adds four documents out of the hard-coded JSON to a *musicians* collection. The Musician model is referenced here to call its create method. `create()` takes one or more documents in JSON form, and a callback to run on completion. If an error occurs, Terminal will spit the error out, and the request will timeout in the browser. On success, 202 Accepted HTTP status code is returned to the browser. Restart your node server and visit this new endpoint to import data.



Returning Data

Now visit your `musicians/` endpoint to view all new musicians data. You'll see an array of JSON objects, each in the defined schema, with an additional generated unique private `_id` and internal `__v` version key. Don't edit those.

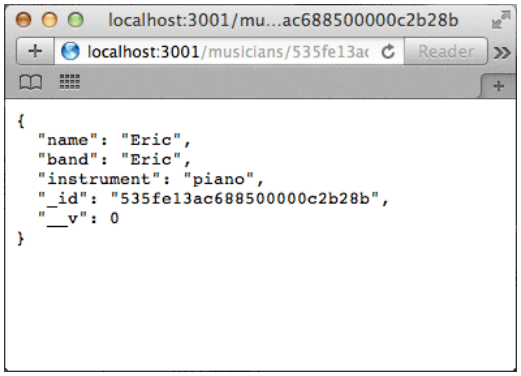


Find By Id

Recall our route for getting a musician by its id `app.get('/musicians/:id', musicians.findById)`. Here is the handler method:

```
1 exports.findById = function(req, res){
2   var id = req.params.id;
3   Musician.findOne({'_id':id},function(err, result) {
4     return res.send(result);
5   });
6 };
```

This route's path uses a parameter pattern for *id* `/musicians/:id` which we can refer to in `req`. Pass this id to Mongoose to look up and return just one document. Restart the server. At your find all endpoint, copy one of the super long ids and paste it in at the end of the current url in the browser. Refresh your browser. You'll get a single JSON object for that one musician's document. Nice.



Update

PUT HTTP actions in a REST API correlate to an Update method. The route for Update also uses an `:id` parameter.

```
1  exports.update = function(req, res) {
2    var id = req.params.id;
3    var updates = req.body;
4
5    Musician.update({"_id":id}, req.body,
6      function (err, numberAffected) {
7        if (err) return console.log(err);
8        console.log('Updated %d musicians', numberAffected);
9        return res.send(202);
10   });
11 }
```

Notice the `updates` variable storing the `req.body`. `req.body` is useful when you want to pass in larger chunks of data such as a single JSON object. Here we will pass in a JSON object following the schema of only the model's properties you want to change.

The model's `update()` takes three parameters:

- JSON object of matching properties to look up the doc with to update
- JSON object of just the properties to update
- callback function that returns the number of documents updated

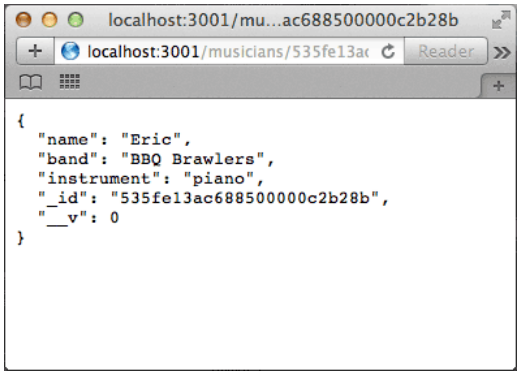
Test with cURL

PUT actions are not easy to test in the browser, so I used cURL in Terminal after restarting the server.

```
1 $ curl -i -X PUT -H 'Content-Type: application/json' -d '{"band": "BBQ Brawlers"}' http://localhost:3001/musicians/535fe13ac688500000c2b28b
```

This sends a JSON Content-Type PUT request to our update endpoint. That JSON object is the request body, and the long hash at the end of the URL is the id of the musician we want to update. Terminal will output a JSON object of the response to the cURL request and *Updated 1 musicians* from our callback function.

Visit this same URL from the cURL request in the browser to see the changes.



Add

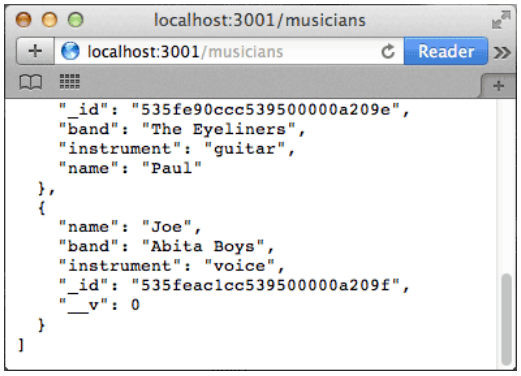
We used `create()` earlier to add multiple documents to our Musicians Mongo collection. Our POST handler uses the same method to add one new Musician to the collection. Once added, the response is the full new Musician's JSON object.

```
1 exports.add = function(req, res) {
2   Musician.create(req.body, function (err, musician) {
3     if (err) return console.log(err);
4     return res.send(musician);
5   });
6 }
```

Restart the server. Use cURL to POST to the add endpoint with the full Musician JSON as the request body.

```
1 $ curl -i -X POST -H 'Content-Type: application/json' -d '{"name": "Joe", "band": "Abita Boys", "instrument":"voice"}' http://localhost:3001/musicians
```

Visit localhost:3001/musicians to see the new Musician at the end of the array.

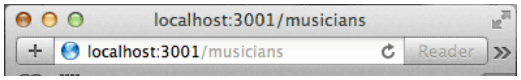


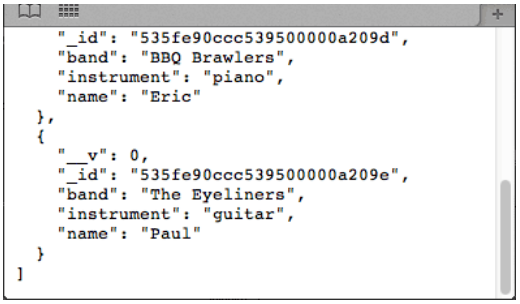
Delete

```
1 exports.delete = function(req, res){
2   var id = req.params.id;
3   Musician.remove({'_id':id},function(result) {
4     return res.send(result);
5   });
6 };
```

Our final REST endpoint, delete reuses what we've learned above. Add this to *controllers/musicians.js*, restart, and check it out with:

```
1 $ curl -i -X DELETE http://localhost:3001/musicians/535feac1cc539500000a209f
```



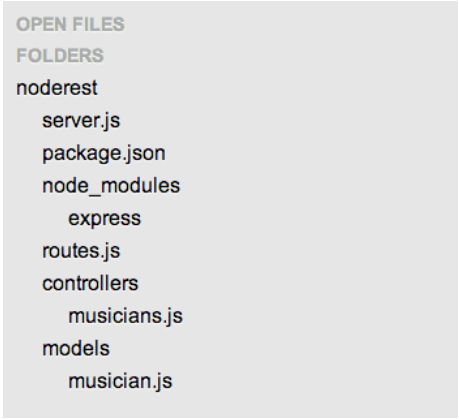


Summary

Your final *controllers/musicians.js* should look like this:

```
1  var mongoose = require('mongoose'),
2  Musician = mongoose.model('Musician');
3
4  exports.findAll = function(req, res){
5    Musician.find({},function(err, results) {
6      return res.send(results);
7    });
8  };
9  exports.findById = function(req, res){
10   var id = req.params.id;
11   Musician.findOne({'_id':id},function(err, result) {
12     return res.send(result);
13   });
14 };
15 exports.add = function(req, res) {
16   Musician.create(req.body, function (err, musician) {
17     if (err) return console.log(err);
18     return res.send(musician);
19   });
20 }
21 exports.update = function(req, res) {
22   var id = req.params.id;
23   var updates = req.body;
24
25   Musician.update({"_id":id}, req.body,
26     function (err, numberAffected) {
27       if (err) return console.log(err);
28       console.log('Updated %d musicians', numberAffected);
29       res.send(202);
30   });
31 }
32 exports.delete = function(req, res){
33   var id = req.params.id;
34   Musician.remove({'_id':id},function(result) {
35     return res.send(result);
36   });
37 };
38
39 exports.import = function(req, res){
40   Musician.create(
41     { "name": "Ben", "band": "DJ Code Red", "instrument": "Reason" },
42     { "name": "Mike D.","band": "Kingston Kats", "instrument": "drums" },
43     { "name": "Eric", "band": "Eric", "instrument": "piano" },
44     { "name": "Paul", "band": "The Eyeliner", "instrument": "guitar" }
45   , function (err) {
46     if (err) return console.log(err);
47     return res.send(202);
48   });
49 };
```

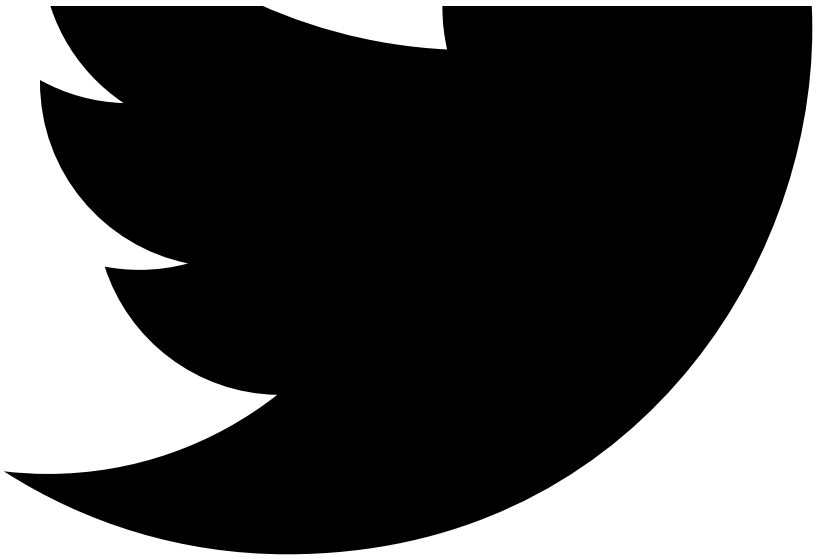
And your project directory should look like this:

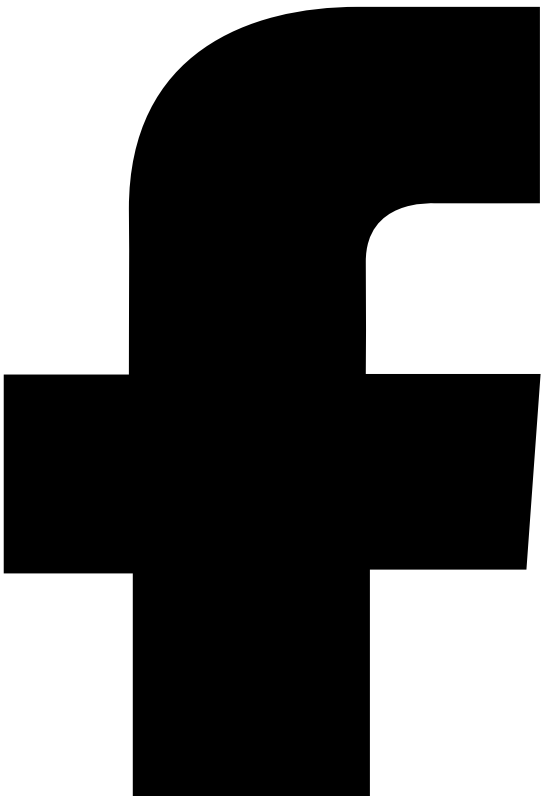


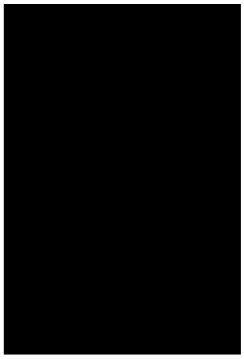
Final project files can be found in the presentation repo.

In my opinion, setting up a REST API with Node.js and Mongo is super fast compared to with PHP and MySQL. The Express.js, Node.js and Mongoose.js API documentation is all easy to reference. There's much more powerful things you can do than I have outlined, so don't stop here.

Shout out to Bespoke.js for the beautiful presentation micro-framework. I'm thinking about adding my fake CLI and fake Directory JS into a Bespoke plugins (arrow through the presentation to see those).







Comments

see more

• Reply • Share ›

Brittney Kernan ➚ Agrawal • 3 years ago

Oo, I have not messed with HBase, could it be a bas64 encoding issue? http://stackoverflow.com/a/100... or the slash at the end of the url? http://support.microsoft.com/k...

• Reply • Share ›

Agrawal • 3 years ago

Thanks Brittney for elaborative article. Its very easy to follow and try out. However I tried CURL command to update the record/row/document and it doesn't update the same. It displays "Updated 0 musicians" from the nodejs server window. The ID I mentioned in the URL is correct and for the first record/row/document. Could you please let me know what's wrong in here? I am using CURL on Windows 7. Printing the value of "req.body" displays "{ '\'{band:BBQ}\'': '' }".

curl -i -X PUT -H 'Content-Type: application/json' -d '{"band":"BBQ"}' http://localhost:3001/musicians/53bba9dbc567ec9c271bf4cd

curl: (6) Could not resolve host: application
HTTP/1.1 202 Accepted
X-Powered-By: Express
Content-Type: text/plain; charset=utf-8
Content-Length: 8
Date: Tue, 08 Jul 2014 15:53:59 GMT
Connection: keep-alive

Accepted

Thanks

-Shesh

• Reply • Share ›

Brittney Kernan ➚ Agrawal • 3 years ago

Hi Agrawal, thank you :). Unfortunately, I'm unable to reproduce your issue on my Mac. I'm pretty sure the problem is with escaping single or double quotes. Please try:

curl -i -X POST -H "Content-Type: application/json" -d "{\"band\":\"BBQ\"}" http://localhost:3001/musicians/53bba9dbc567ec9c271bf4cd

Did you try '{\"band\":\"BBQ\"}'? Maybe "{\"band\":\"BBQ\"}" or [{\"band\":\"BBQ\"}]? Did you review that stackoverflow page I linked to below? I'd google more 'windows curl json' articles and give answers provided a try.

• Reply • Share ›

Agrawal • 3 years ago

Further if I print the value of req.body in
"exports.add = function(req, res)" (musicians.js) then it prints as follows:

{ '\'{name:Shesh1,band:BBQ,instrument:flute}\'': '' }

For command line: curl -i -X POST -d '{"name":"Shesh1","band":"BBQ","instrument":"flute"}' http://localhost:3001/musicians

However if I hard code the value to be inserted as follows, then it inserts successfully. Could you please let me know what's wrong in the way I am passing the value from curl?

Musician.create({ "name": "Shesh", "band": "Indian", "instrument": "Flute" }, ...
Any help in this regard is highly appreciated.
Thanks

• Reply • Share ›

Agrawal • 3 years ago

I also tried little modified curl command, it doesn't report any error but at the same time it doesn't update record/row/document in MongoDB and node server terminal continues to display "Updated 0 musicians" ??

curl -i -X PUT -d '{"band":"BBQ Brawlers"}' http://localhost:3001/musicians/53bba9dbc567ec9c271bf4cd

HTTP/1.1 202 Accepted
X-Powered-By: Express
Content-Type: text/plain; charset=utf-8
Content-Length: 8
Date: Tue, 08 Jul 2014 11:06:16 GMT
Connection: keep-alive

Accepted

• Reply • Share ›

Agrawal • 3 years ago

When I try to update the record/row/document using the curl (on Windows), it says "Updated 0 musicians" on node server terminal. I checked that I am providing correct ID for a record/row/document. What could be wrong?

curl -i -X PUT -H 'Content-Type: application/json' -d '[{"band": "BBQ Brawlers"}]' http://localhost:3001/musicians/53bba9dbc567ec9c271bf4cd

curl: (6) Could not resolve host: application
curl: (3) [globbing] unmatched close brace/bracket in column 13
HTTP/1.1 202 Accepted
X-Powered-By: Express
Content-Type: text/plain; charset=utf-8
Content-Length: 8
Date: Tue, 08 Jul 2014 09:57:49 GMT
Connection: keep-alive

Reply • Share ›

Brittney Kernan ➚ Agrawal • 3 years ago

Hi Agrawal! Thanks reading.

With your windows+cURL set up you may have to escape all of the double quotes as mentioned here http://stackoverflow.com/quest.... Have you tried that?

It may also be your version of cURL. My version is: curl 7.24.0 for Mac with OpenSSL (this may be what you need).

Reply • Share ›

Agrawal • 3 years ago

Thanks Brittney, Indeed good explanation and its very simple to follow.

Reply • Share ›

Subscribe Add Disqus to your site Add Disqus Add Privacy

Check Out More:

Front-End Dev Software List

By Paul Graffam

Hello, World

By Ian Solano

We are a different kind of digital agency. We bring together product design, brand communications, social connections and content to help businesses thrive.

Careers

Email

RSS