[Christophe Coenraets](http://coenraets.org/blog/) Web Platform, Cloud and Mobile Application Development

Top of Form



Bottom of Form

* [Home](http://coenraets.org)
* [Cordova PhoneGap Tutorial](http://ccoenraets.github.io/cordova-tutorial/)
* [Salesforce Developer Tutorial](http://ccoenraets.github.io/salesforce-developer-workshop/)
* [Bio](http://coenraets.org/blog/bio/)

**Creating a REST API using Node.js, Express, and MongoDB**

By [Christophe Coenraets](http://coenraets.org/blog/author/admin/) on October 2, 2012 in [Express](http://coenraets.org/blog/category/express/), [JavaScript](http://coenraets.org/blog/category/javascript/), [MongoDB](http://coenraets.org/blog/category/mongodb/), [Node.js](http://coenraets.org/blog/category/node-js/), [REST](http://coenraets.org/blog/category/rest/)

[](http://coenraets.org/blog/wp-content/uploads/2012/10/nodemango1.jpg)  
I recently used Node.js, Express, and MongoDB to rewrite a RESTful API I had previously written in Java and PHP with MySQL ([Java version](http://coenraets.org/blog/2011/12/restful-services-with-jquery-and-java-using-jax-rs-and-jersey/), [PHP version](http://coenraets.org/blog/2011/12/restful-services-with-jquery-php-and-the-slim-framework/)), and I thought I’d share the experience…

Here is a quick guide showing how to build a RESTful API using [Node.js](http://nodejs.org/), [Express](http://expressjs.com/), and [MongoDB](http://www.mongodb.org/).

**Installing Node.js**

1. Go to <http://nodejs.org>, and click the *Install* button.
2. Run the installer that you just downloaded. When the installer completes, a message indicates that *Node was installed at /usr/local/bin/node* and *npm was installed at /usr/local/bin/npm*.

At this point node.js is ready to use. Let’s implement the webserver application from the nodejs.org home page. We will use it as a starting point for our project: a RESTful API to access data (retrieve, create, update, delete) in a wine cellar database.

1. Create a folder named *nodecellar* anywhere on your file system.
2. In the wincellar folder, create a file named *server.js*.
3. Code server.js as follows:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | var http = require('http');  http.createServer(function (req, res) {  res.writeHead(200, {'Content-Type': 'text/plain'});  res.end('Hello World\n');  }).listen(3000, '127.0.0.1');  console.log('Server running at http://127.0.0.1:3000/'); |

1. [view raw](https://gist.github.com/ccoenraets/3815873/raw/server.js) [server.js](https://gist.github.com/ccoenraets/3815873#file-server-js) hosted with ❤ by [GitHub](https://github.com)

We are now ready to start the server and test the application:

1. To start the server, open a shell, *cd* to your nodecellar directory, and start your server as follows:

node server.js

1. To test the application, open a browser and access <http://localhost:3000>.

**Installing Express**

Express is a lightweight node.js web application framework. It provides the basic HTTP infrastructure that makes it easy to create REST APIs.

To install Express in the nodecellar application:

1. In the nodecellar folder, create a file named *package.json* defined as follows:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | {  "name": "wine-cellar",  "description": "Wine Cellar Application",  "version": "0.0.1",  "private": true,  "dependencies": {  "express": "3.x"  }  } |

1. [view raw](https://gist.github.com/ccoenraets/3815974/raw/package.json) [package.json](https://gist.github.com/ccoenraets/3815974#file-package-json) hosted with ❤ by [GitHub](https://github.com)
2. Open a shell, *cd* to the nodecellar directory, and execute the following command to install the express module.

npm install

A *node\_modules* folder is created in the nodecellar folder, and the Express module is installed in a subfolder of node\_modules.

Now that Express is installed, we can stub a basic REST API for the nodecellar application:

1. Open server.js and replace its content as follows:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | var express = require('express');    var app = express();    app.get('/wines', function(req, res) {  res.send([{name:'wine1'}, {name:'wine2'}]);  });  app.get('/wines/:id', function(req, res) {  res.send({id:req.params.id, name: "The Name", description: "description"});  });    app.listen(3000);  console.log('Listening on port 3000...'); |

1. [view raw](https://gist.github.com/ccoenraets/3815974/raw/server.js) [server.js](https://gist.github.com/ccoenraets/3815974#file-server-js) hosted with ❤ by [GitHub](https://github.com)
2. Stop (CTRL+C) and restart the server:

node server

1. To test the API, open a browser and access the following URLs:

|  |  |
| --- | --- |
| Get all the wines in the database: | <http://localhost:3000/wines> |
| Get wine with a specific id (for example: 1): | <http://localhost:3000/wines/1> |

**Using Node.js Modules**

In a large application, things could easily get out of control if we keep adding code to a single JavaScript file (server.js). Let’s move the wine-related code in a *wines* module that we then declare as a dependency in server.js.

1. In the nodecellar folder, create a subfolder called *routes*.
2. In the routes folder create a file named *wines.js* and defined as follows:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | exports.findAll = function(req, res) {  res.send([{name:'wine1'}, {name:'wine2'}, {name:'wine3'}]);  };    exports.findById = function(req, res) {  res.send({id:req.params.id, name: "The Name", description: "description"});  }; |

1. [view raw](https://gist.github.com/ccoenraets/3816103/raw/wines.js) [wines.js](https://gist.github.com/ccoenraets/3816103#file-wines-js) hosted with ❤ by [GitHub](https://github.com)
2. Modify server.js as follows to delegate the routes implementation to the wines module:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | var express = require('express'),  wines = require('./routes/wines');    var app = express();    app.get('/wines', wines.findAll);  app.get('/wines/:id', wines.findById);    app.listen(3000);  console.log('Listening on port 3000...'); |

1. [view raw](https://gist.github.com/ccoenraets/3816103/raw/server.js) [server.js](https://gist.github.com/ccoenraets/3816103#file-server-js) hosted with ❤ by [GitHub](https://github.com)
2. Restart the server and test the APIs:

|  |  |
| --- | --- |
| Get all the wines in the database: | <http://localhost:3000/wines> |
| Get wine with a specific id (for example: 1): | <http://localhost:3000/wines/1> |

The next step is to replace the placeholder data with actual data from a MongoDB database.

**Installing MongoDB**

To install MongoDB on your specific platform, refer to the [MongoDB QuickStart](http://www.mongodb.org/display/DOCS/Quickstart). Here are some quick steps to install MongoDB on a Mac:

1. Open a terminal window and type the following command to download the latest release:

curl http://downloads.mongodb.org/osx/mongodb-osx-x86\_64-2.2.0.tgz > ~/Downloads/mongo.tgz

Note: You may need to adjust the version number. 2.2.0 is the latest production version at the time of this writing.

1. Extract the files from the mongo.tgz archive:

cd ~/Downloads  
tar -zxvf mongo.tgz

1. Move the mongo folder to /usr/local (or another folder according to your personal preferences):

sudo mv -n mongodb-osx-x86\_64-2.2.0/ /usr/local/

1. (Optional) Create a symbolic link to make it easier to access:

sudo ln -s /usr/local/mongodb-osx-x86\_64-2.2.0 /usr/local/mongodb

1. Create a folder for MongoDB’s data and set the appropriate permissions:

sudo mkdir -p /data/db  
sudo chown `id -u` /data/db

1. Start mongodb

cd /usr/local/mongodb  
./bin/mongod

1. You can also open the MongoDB Interactive Shell in another terminal window to interact with your database using a command line interface.

cd /usr/local/mongodb  
./bin/mongo

Refer to the [MongoDB Interactive Shell documentation](http://www.mongodb.org/display/DOCS/Overview+-+The+MongoDB+Interactive+Shell) for more information.

**Installing the MongoDB Driver for Node.js**

There are different solutions offering different levels of abstraction to access MongoDB from Node.js (For example, [Mongoose](http://mongoosejs.com/) and [Mongolia](https://github.com/masylum/mongolia)). A comparaison of these solutions is beyond the scope of this article. In this, guide we use the [native Node.js driver](http://www.mongodb.org/display/DOCS/node.JS).

To install the the native Node.js driver, open a terminal window, cd to your nodecellar folder, and execute the following command:

npm install mongodb

**Implementing the REST API**

The full REST API for the nodecellar application consists of the following methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **URL** | **Action** |
| GET | /wines | Retrieve all wines |
| GET | /wines/5069b47aa892630aae000001 | Retrieve the wine with the specified \_id |
| POST | /wines | Add a new wine |
| PUT | /wines/5069b47aa892630aae000001 | Update wine with the specified \_id |
| DELETE | /wines/5069b47aa892630aae000001 | Delete the wine with the specified \_id |

To implement all the *routes* required by the API, modify server.js as follows:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | var express = require('express'),  wine = require('./routes/wines');    var app = express();    app.configure(function () {  app.use(express.logger('dev')); /\* 'default', 'short', 'tiny', 'dev' \*/  app.use(express.bodyParser());  });    app.get('/wines', wine.findAll);  app.get('/wines/:id', wine.findById);  app.post('/wines', wine.addWine);  app.put('/wines/:id', wine.updateWine);  app.delete('/wines/:id', wine.deleteWine);    app.listen(3000);  console.log('Listening on port 3000...'); |

[view raw](https://gist.github.com/ccoenraets/3819468/raw/server.js) [server.js](https://gist.github.com/ccoenraets/3819468#file-server-js) hosted with ❤ by [GitHub](https://github.com)

To provide the data access logic for each route, modify wines.js as follows:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 | var mongo = require('mongodb');    var Server = mongo.Server,  Db = mongo.Db,  BSON = mongo.BSONPure;    var server = new Server('localhost', 27017, {auto\_reconnect: true});  db = new Db('winedb', server);    db.open(function(err, db) {  if(!err) {  console.log("Connected to 'winedb' database");  db.collection('wines', {strict:true}, function(err, collection) {  if (err) {  console.log("The 'wines' collection doesn't exist. Creating it with sample data...");  populateDB();  }  });  }  });    exports.findById = function(req, res) {  var id = req.params.id;  console.log('Retrieving wine: ' + id);  db.collection('wines', function(err, collection) {  collection.findOne({'\_id':new BSON.ObjectID(id)}, function(err, item) {  res.send(item);  });  });  };    exports.findAll = function(req, res) {  db.collection('wines', function(err, collection) {  collection.find().toArray(function(err, items) {  res.send(items);  });  });  };    exports.addWine = function(req, res) {  var wine = req.body;  console.log('Adding wine: ' + JSON.stringify(wine));  db.collection('wines', function(err, collection) {  collection.insert(wine, {safe:true}, function(err, result) {  if (err) {  res.send({'error':'An error has occurred'});  } else {  console.log('Success: ' + JSON.stringify(result[0]));  res.send(result[0]);  }  });  });  }    exports.updateWine = function(req, res) {  var id = req.params.id;  var wine = req.body;  console.log('Updating wine: ' + id);  console.log(JSON.stringify(wine));  db.collection('wines', function(err, collection) {  collection.update({'\_id':new BSON.ObjectID(id)}, wine, {safe:true}, function(err, result) {  if (err) {  console.log('Error updating wine: ' + err);  res.send({'error':'An error has occurred'});  } else {  console.log('' + result + ' document(s) updated');  res.send(wine);  }  });  });  }    exports.deleteWine = function(req, res) {  var id = req.params.id;  console.log('Deleting wine: ' + id);  db.collection('wines', function(err, collection) {  collection.remove({'\_id':new BSON.ObjectID(id)}, {safe:true}, function(err, result) {  if (err) {  res.send({'error':'An error has occurred - ' + err});  } else {  console.log('' + result + ' document(s) deleted');  res.send(req.body);  }  });  });  }    /\*--------------------------------------------------------------------------------------------------------------------\*/  // Populate database with sample data -- Only used once: the first time the application is started.  // You'd typically not find this code in a real-life app, since the database would already exist.  var populateDB = function() {    var wines = [  {  name: "CHATEAU DE SAINT COSME",  year: "2009",  grapes: "Grenache / Syrah",  country: "France",  region: "Southern Rhone",  description: "The aromas of fruit and spice...",  picture: "saint\_cosme.jpg"  },  {  name: "LAN RIOJA CRIANZA",  year: "2006",  grapes: "Tempranillo",  country: "Spain",  region: "Rioja",  description: "A resurgence of interest in boutique vineyards...",  picture: "lan\_rioja.jpg"  }];    db.collection('wines', function(err, collection) {  collection.insert(wines, {safe:true}, function(err, result) {});  });    }; |

[view raw](https://gist.github.com/ccoenraets/3819468/raw/wines.js) [wines.js](https://gist.github.com/ccoenraets/3819468#file-wines-js) hosted with ❤ by [GitHub](https://github.com)

Restart the server to test the API.

**Testing the API using cURL**

If you want to test your API before using it in a client application, you can invoke your REST services straight from a browser address bar. For example, you could try:

* <http://localhost:3000/wines>

You will only be able to test your GET services that way. A more versatile solution to test RESTful services is to use [cURL](http://curl.haxx.se/), a command line utility for transferring data with URL syntax.

For example, using cURL, you can test the Wine Cellar API with the following commands:

* Get all wines:

curl -i -X GET http://localhost:3000/wines

* Get wine with \_id value of 5069b47aa892630aae000007 (use a value that exists in your database):

curl -i -X GET http://localhost:3000/wines/5069b47aa892630aae000007

* Delete wine with \_id value of 5069b47aa892630aae000007:

curl -i -X DELETE http://localhost:3000/wines/5069b47aa892630aae000007

* Add a new wine:

curl -i -X POST -H 'Content-Type: application/json' -d '{"name": "New Wine", "year": "2009"}' http://localhost:3000/wines

* Modify wine with \_id value of 5069b47aa892630aae000007:

curl -i -X PUT -H 'Content-Type: application/json' -d '{"name": "New Wine", "year": "2010"}' http://localhost:3000/wines/5069b47aa892630aae000007