



Technologies > Web Development

Creating RESTful APIs with NodeJS and MongoDB Tutorial (Part II)


 Last update October 14th 2016  482.4k 75 Comments
 [#expressjs](#) [#mongodb](#) [#nodejs](#) [#tutorial_mean-stack](#) [#javascript](#)



Welcome to this tutorial about RESTful API using Node.js (Express.js) and MongoDB (mongoose)! We are going to learn how to install and use each component individually and then proceed to create a RESTful API.

MEAN Stack tutorial series:

SHARES

2. Creating RESTful APIs with NodeJS and MongoDB Tutorial (Part II)  **you are here**
3. [MEAN Stack Tutorial: MongoDB, ExpressJS, AngularJS and NodeJS \(Part III\)](#)

1. What is a RESTful API?

REST stands for Representational State Transfer. It is an architecture that allows client-server communication through a uniform interface. REST is `stateless` , `cachable` and has property called `idempotence` . It means that the side effect of identical requests have the same side-effect as a single request.

HTTP RESTful API's are compose of:

- HTTP methods, e.g. GET, PUT, DELETE, PATCH, POST, ...
- Base URI, e.g. `http://adrianmejia.com`
- URL path, e.g. `/blog/2014/10/01/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/`
- Media type, e.g. `html` , `JSON` , `XML` , `Microformats` , `Atom` , `Images` ...

Here is a summary what we want to implement:

Resource (URI)	POST (create)	GET (read)	PUT (update)	DELETE (destroy)
/todos	create new task	list tasks	N/A (update all)	N/A (destroy all)
/todos/1	error	show task ID 1	update task ID 1	destroy task ID 1

NOTE for this tutorial:

- Format will be JSON.
- Bulk updates and bulk destroys are not safe, so we will not be implementing those.

SHARES

- **CRUD** functionality: POST == **CREATE**, GET == **READ**, PUT == **UPDATE**, DELETE == **DELETE**.

2. Installing the MEAN Stack Backend

In this section, we are going to install the backend components of the MEAN stack: MongoDB, NodeJS and ExpressJS. If you already are familiar with them, then jump to [wiring the stack](#). Otherwise, enjoy the ride!

2.1 Installing MongoDB

MongoDB is a document-oriented NoSQL database (Big Data ready). It stores data in JSON-like format and allows users to perform SQL-like queries against it.

You can install MongoDB following the [instructions here](#).

If you have a **Mac** and [brew](#) it's just:

```
1 brew install mongodb && mongod
```

In **Ubuntu**:

```
1 sudo apt-get -y install mongodb
```

After you have them installed, check version as follows:

```
1 # Mac
2 mongod --version
```

SHARES

```
5
6 # Ubuntu
7 mongod --version
8 # => db version v2.0.4, pdfile version 4.5
9 # => Wed Oct 1 23:06:54 git version: nogitversion
```

2.2 Installing NodeJS

The Node official definition is:

“Node.js® is a JavaScript runtime built on Chrome’s V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js’ package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Node.js Website nodejs.org

In short, NodeJS allows you to run Javascript outside the browser, in this case, on the web server. NPM allows you to install/publish node packages with ease.

To install it, you can go to the [NodeJS Website](https://nodejs.org).

Since Node versions changes very often. You can use the NVM (Node Version Manager) on **Ubuntu** and Mac with:

```
1 # download NPM
2 curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.4/ir
3
4 # load NPM
5 export NVM_DIR="$HOME/.nvm"
6 [ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh" # This loads nvm
7
8 # Install latest stable version
9 nvm install stable
```

Check out <https://github.com/creationix/nvm> for more details

SHARES

Also, on **Mac** and [brew](#) you can do:

```
1 brew install nodejs
```

After you got it installed, check node version and npm (node package manager) version:

```
1 node -v
2 # => v6.2.2
3
4 npm -v
5 # => 3.9.5
```

2.3 Installing ExpressJS

ExpressJS is a web application framework that runs on NodeJS. It allows you to build web applications and API endpoints. (more details on this later).

We are going to create a project folder first, and then add `express` as a dependency. Let's use NPM `init` command to get us started.

```
1 # create project folder
2 mkdir todo-app
3
4 # move to the folder and initialize the project
5 cd todo-app
6 npm init . # press enter multiple times to accept all defaults
7
8 # install express v4.14 and save it as dependency
9 npm install express@4.14 --save
```

Notice that after the last command, `express` should be added to `package.json` with the version `4.14.x`.

SHARES

3. Using MongoDB with Mongoose

Mongoose is an NPM package that allows you to interact with MongoDB. You can install it as follows:

```
1 npm install mongoose@4.5.8 --save
```

If you followed the previous steps, you should have all you need to complete this tutorial. We are going to build an API that allow users to CRUD (Create-Read-Update-Delete) Todo tasks from database.

3.1 Mongoose CRUD

CRUD == **C**reate-**R**ead-**U**ppdate-**D**eleate

We are going to create, read, update and delete data from MongoDB using Mongoose/Node. First, you need to have mongod up and running:

```
1 # run mongo daemon
2 mongod
```

Keep mongo running in a terminal window and while in the folder `todoApp` type `node` to enter the node CLI. Then:

```
1 // Load mongoose package
2 var mongoose = require('mongoose');
3
4 // Connect to MongoDB and create/use database called todoAppTest
5 mongoose.connect('mongodb://localhost/todoAppTest');
```

SHARES

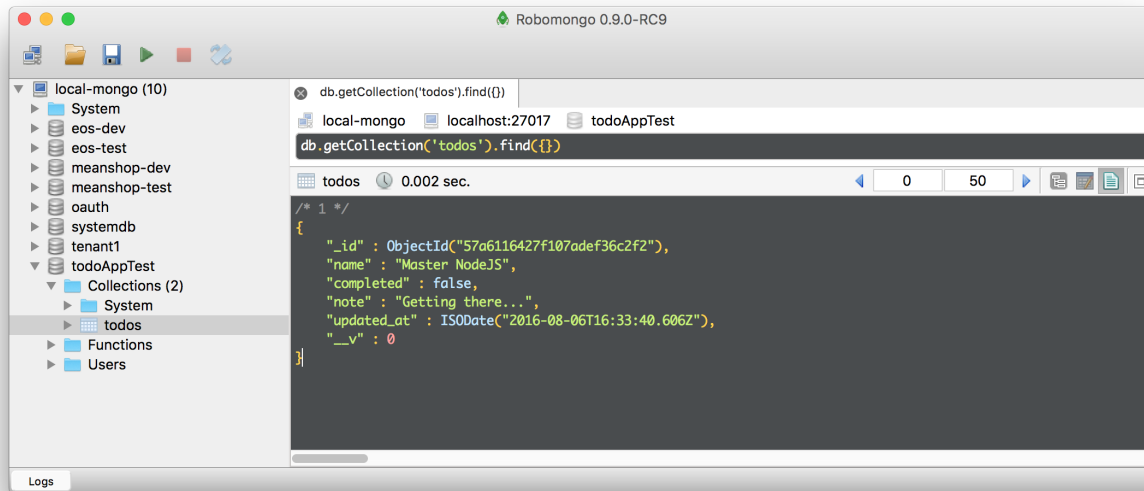
```
8  var TodoSchema = new mongoose.Schema({
9    name: String,
10   completed: Boolean,
11   note: String,
12   updated_at: { type: Date, default: Date.now },
13 });
14
15 // Create a model based on the schema
16 var Todo = mongoose.model('Todo', TodoSchema);
```

Great! Now, let's test that we can save and edit data.

3.2 Mongoose Create

```
1
2 // Create a todo in memory
3 var todo = new Todo({name: 'Master NodeJS', completed: false, note:
4
5 // Save it to database
6 todo.save(function(err){
7   if(err)
8     console.log(err);
9   else
10     console.log(todo);
11 });
```

If you take a look to Mongo you will notice that we just created an entry. You can easily visualize data using [Robomongo](#):



You can also build the object and save it in one step using `create` :

```
1 Todo.create({name: 'Create something with Mongoose', completed: true
2   if(err) console.log(err);
3   else console.log(todo);
4 });
```

3.3 Mongoose Read and Query

So far we have been able to save data, now we are going to explore how to query the information. There are multiple options for reading/querying data:

- `Model.find(conditions, [fields], [options], [callback])`
- `Model.findById(id, [fields], [options], [callback])`
- `Model.findOne(conditions, [fields], [options], [callback])`

Some examples:

Find all

```
1 // Find all data in the Todo collection
2 Todo.find(function (err, todos) {
```

SHARES


```
5  });
```

The result is something like this:

results

```
1  [ { _id: 57a6116427f107adef36c2f2,
2      name: 'Master NodeJS',
3      completed: false,
4      note: 'Getting there...',
5      __v: 0,
6      updated_at: 2016-08-06T16:33:40.606Z },
7    { _id: 57a6142127f107adef36c2f3,
8      name: 'Create something with Mongoose',
9      completed: true,
10     note: 'this is one',
11     __v: 0,
12     updated_at: 2016-08-06T16:45:21.143Z } ]
```

You can also add queries

Find with queries

```
1  // callback function to avoid duplicating it all over
2  var callback = function (err, data) {
3    if (err) { return console.error(err); }
4    else { console.log(data); }
5  }
6
7  // Get ONLY completed tasks
8  Todo.find({completed: true }, callback);
9
10 // Get all tasks ending with `JS`
11 Todo.find({name: /JS$/ }, callback);
```

You can chain multiple queries, e.g.:

Chaining queries

SHARES

```
3 // Get all tasks staring with `Master`, completed
4 Todo.find({name: /^Master/, completed: true }, callback);
5
6 // Get all tasks staring with `Master`, not completed and created fr
7 Todo.find({name: /^Master/, completed: false }).where('updated_at').
8
```

MongoDB query language is very powerful. We can combine regular expressions, date comparison and more!

3.4 Mongoose Update

Moving on, we are now going to explore how to update data.

Each model has an `update` method which accepts multiple updates (for batch updates, because it doesn't return an array with data).

- `Model.update(conditions, update, [options], [callback])`
- `Model.findByIdAndUpdate(id, [update], [options], [callback])`
- `Model.findOneAndUpdate([conditions], [update], [options], [callback])`

Alternatively, the method `findOneAndUpdate` could be used to update just one and return an object.

`Todo.update` and `Todo.findOneAndUpdate`

```
1
2 // Model.update(conditions, update, [options], [callback])
3 // update `multi`ple tasks from complete false to true
4
5 Todo.update({ name: /master/i }, { completed: true }, { multi: true
6
7 //Model.findOneAndUpdate([conditions], [update], [options], [callback])
8 Todo.findOneAndUpdate({name: /JS$/ }, {completed: false}, callback);
```

As you might noticed the batch updates (`multi: true`) doesn't show the data, rather

SHARES

```
1 { ok: 1, nModified: 1, n: 1 }
```

Here is what they mean:

- `n` means the number of records that matches the query
- `nModified` represents the number of documents that were modified with update query.

3.5 Mongoose Delete

`update` and `remove` mongoose API are identical, the only difference it is that no elements are returned. Try it on your own ;)

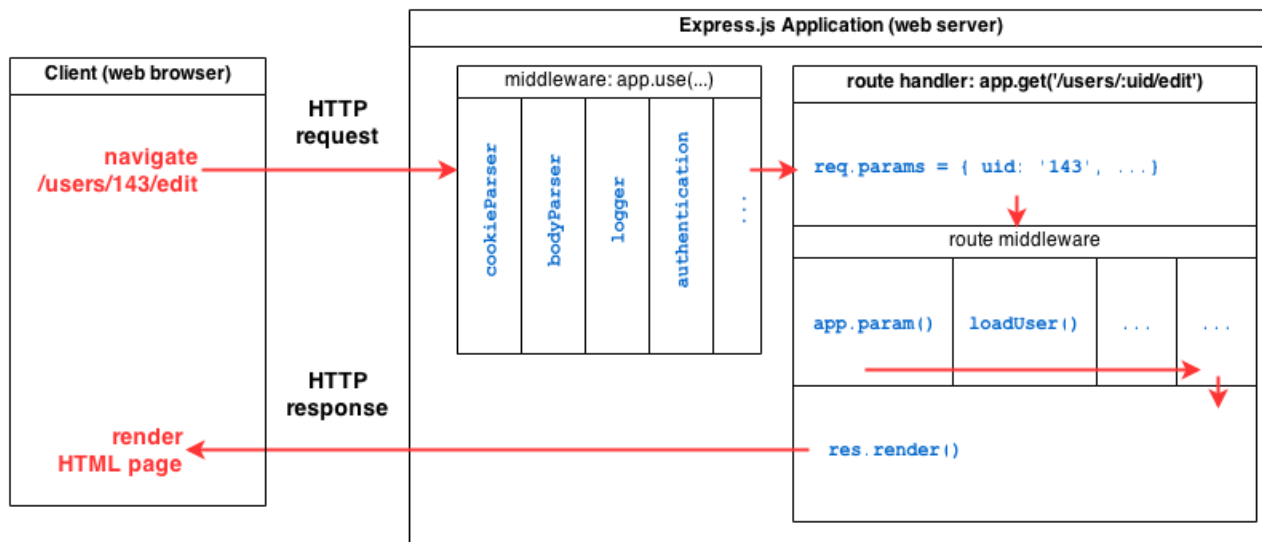
- `Model.remove(conditions, [callback])`
- `Model.findByIdAndRemove(id, [options], [callback])`
- `Model.findOneAndRemove(conditions, [options], [callback])`

4. ExpressJS and Middlewares

ExpressJS is a complete web framework solution. It has HTML template solutions (jade, ejs, handlebars, hogan.js) and CSS precompilers (less, stylus, compass). Through middlewares layers, it handles: cookies, sessions, caching, CSRF, compression and many more.

Middlewares are pluggable processors that runs on each request made to the server. You can have any number of middlewares that will process the request one by one in a serial fashion. Some middlewares might alter the request input. Others, might create log outputs, add data and pass it to the `next()` middleware in the chain.

We can use the middlewares using `app.use` . That will apply for all request. If you want to be more specific, you can use `app.verb` . For instance: `app.get`, `app.delete`, `app.post`, `app.update`, ...



Let's give some examples of middlewares to drive the point home.

Say you want to log the IP of the client on each request:

Log the client IP on every request

```
1 app.use(function (req, res, next) {  
2   var ip = req.headers['x-forwarded-for'] || req.connection.remoteAddress  
3   console.log('Client IP:', ip);  
4   next();  
5 });
```

Notice that each middleware has 3 parameters:

- `req` : contain all the requests objects like URLs, path, ...
- `res` : is the response object where we can send the reply back to the client.
- `next` : continue with the next middleware in the chain.

You can also specify a path that you want the middleware to activate on.

Middleware mounted on `"/todos/:id"` and log the request method

SHARES

```
3   next();  
4 });
```

And finally you can use `app.get` to catch GET requests with matching routes, reply the request with a `response.send` and end the middleware chain. Let's use what we learned on [mongoose read](#) to reply with the user's data that matches the `id`.

Middleware mounted on `"/todos/:id"` and returns

```
1 app.get('/todos/:id', function (req, res, next) {  
2   Todo.findById(req.params.id, function(err, todo){  
3     if(err) res.send(err);  
4     res.json(todo);  
5   });  
6 });
```

Notice that all previous middlewares called `next()` except this last one, because it sends a response (in JSON) to the client with the requested `todo` data.

Hopefully, you don't have to develop a bunch of middlewares besides routes, since ExpressJS has a bunch of middlewares available.

4.1 Default Express 4.0 middlewares

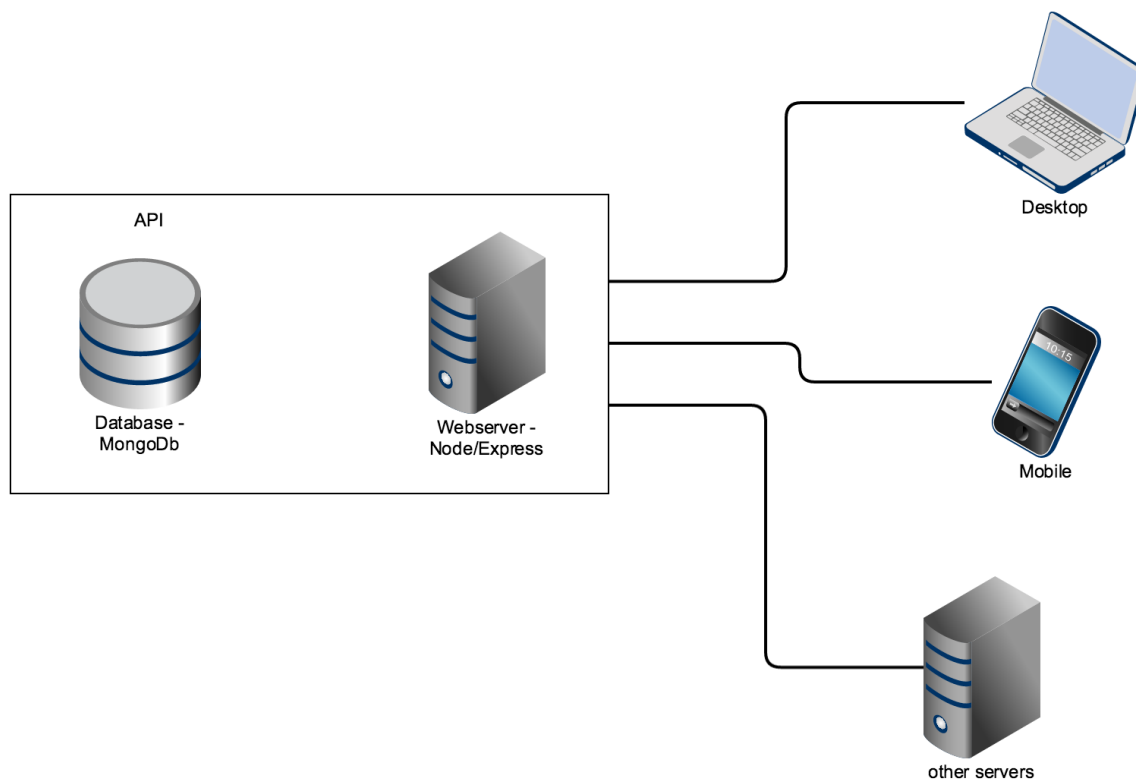
- [morgan](#): logger
- [body-parser](#): parse the body so you can access parameters in requests in `req.body` . e.g. `req.body.name` .
- [cookie-parser](#): parse the cookies so you can access parameters in cookies `req.cookies` . e.g. `req.cookies.name` .
- [serve-favicon](#): exactly that, serve favicon from route `/favicon.ico` . Should be call on the top before any other routing/middleware takes place to avoids unnecessary parsing.

The following middlewares are not added by default, but it's nice to know they exist at least:

- **compression**: compress all request. e.g. `app.use(compression())`
- **session**: create sessions. e.g. `app.use(session({secret: 'Secr3t'}))`
- **method-override**: `app.use(methodOverride('_method'))` Override methods to the one specified on the `_method` param. e.g. `GET /resource/1?_method=DELETE` will become `DELETE /resource/1`.
- **response-time**: `app.use(responseTime())` adds `X-Response-Time` header to responses.
- **errorhandler**: Aid development, by sending full error stack traces to the client when an error occurs. `app.use(errorhandler())`. It is good practice to surround it with an if statement to check `process.env.NODE_ENV === 'development'`.
- **vhost**: Allows you to use different stack of middlewares depending on the request hostname. e.g. `app.use(vhost('*.user.local', userapp))` and `app.use(vhost('assets-*.example.com', staticapp))` where `userapp` and `staticapp` are different express instances with different middlewares.
- **csrf**: Adds a **C**ross-site **r**quest **f**orgery (CSRF) protection by adding a token to responds either via `session` or `cookie-parser` middleware.
`app.use(csrf());`
- **timeout**: halt execution if it takes more that a given time. e.g.
`app.use(timeout('5s'))`; . However you need to check by yourself under every request with a middleware that checks `if (!req.timedout) next();`.

5. Wiring up the MEAN Stack

In the next sections, we are going to put together everything that we learn from and build an API. They can be consumed by browsers, mobile apps and even other servers.



5.1 Bootstrapping ExpressJS

After a detour in the land of Node, MongoDB, Mongoose, and middlewares, we are back to our express todoApp. This time to create the routes and finalize our RESTful API.

Express has a separate package called `express-generator`, which can help us to get started with our API.

Install and run "express-generator"

```
1 # install it globally using -g
2 npm install express-generator -g
3
4 # create todo-app API with EJS views (instead the default Jade)
5 express todo-api -e
```

SHARES

```
8 # create : todo-api/package.json
9 # create : todo-api/app.js
10 # create : todo-api/public
11 # create : todo-api/public/javascripts
12 # create : todo-api/routes
13 # create : todo-api/routes/index.js
14 # create : todo-api/routes/users.js
15 # create : todo-api/public/stylesheets
16 # create : todo-api/public/stylesheets/style.css
17 # create : todo-api/views
18 # create : todo-api/views/index.ejs
19 # create : todo-api/views/layout.ejs
20 # create : todo-api/views/error.ejs
21 # create : todo-api/public/images
22 # create : todo-api/bin
23 # create : todo-api/bin/www
24 #
25 # install dependencies:
26 #   $ cd todo-api && npm install
27 #
28 # run the app on Linux/Mac:
29 #   $ DEBUG=todo-app:* npm start
30 #
31 # run the app on Windows:
32 #   $ SET DEBUG=todo-api:* & npm start
```

This will create a new folder called `todo-api` . Let's go ahead and install the dependencies and run the app:

```
1 # install dependencies
2 cd todo-api && npm install
3
4 # run the app on Linux/Mac
5 PORT=4000 npm start
6
7 # run the app on Windows
8 SET PORT=4000 & npm start
```

Use your browser to go to <http://0.0.0.0:4000>, and you should see a message “Welcome

SHARES

5.2 Connect ExpressJS to MongoDB

In this section we are going to access MongoDB using our newly created express app. Hopefully, you have installed MongoDB in the [setup section](#), and you can start it by typing (if you haven't yet):

```
1  mongod
```

Install the MongoDB driver for NodeJS called mongoose:

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

Notice `--save` . It will add it to the `todo-api/package.json`

Next, you need to require mongoose in the `todo-api/app.js`

Add to app.js

```
1  // load mongoose package
2  var mongoose = require('mongoose');
3
4  // Use native Node promises
5  mongoose.Promise = global.Promise;
6
7  // connect to MongoDB
8  mongoose.connect('mongodb://localhost/todo-api')
9    .then(() => console.log('connection succesful'))
10   .catch((err) => console.error(err));
```

Now, When you run `npm start` or `./bin/www` , you will notice the message `connection successful` . Great!

You can find the repository [here](#) and the diff code at this point: [diff](#)

5.3 Creating the Todo model with Mongoose

It's show time! All the above was setup and preparation for this moment. Let bring the API to life.

Create a `models` directory and a `Todo.js` model:

```
1 mkdir models
2 touch models/Todo.js
```

In the `models/Todo.js` :

```
1 var mongoose = require('mongoose');
2
3 var TodoSchema = new mongoose.Schema({
4   name: String,
5   completed: Boolean,
6   note: String,
7   updated_at: { type: Date, default: Date.now },
8 });
9
10 module.exports = mongoose.model('Todo', TodoSchema);
```

diff

What's going on up there? Isn't MongoDB suppose to be schemaless? Well, it is schemaless and very flexible indeed. However, very often we want bring sanity to our API/WebApp through validations and enforcing a schema to keep a consistent structure. Mongoose does that for us, which is nice.

You can use the following types:

- String
- Boolean

SHARES

- Array
- Number
- ObjectId
- Mixed
- Buffer

6. API clients (Browser, Postman and curl)

I know you have not created any route yet. However, in the next sections you will. These are just three ways to retrieve, change and delete data from your future API.

6.1 Curl

Create tasks

```
1 # Create task
2 curl -XPOST http://localhost:3000/todos -d 'name=Master%20Routes&con
3
4 # List tasks
5 curl -XGET http://localhost:3000/todos
```

6.2 Browser and Postman

If you open your browser and type `localhost:3000/todos` you will see all the tasks (when you implement it). However, you cannot do post commands by default. For further testing let's use a Chrome plugin called [Postman](#). It allows you to use all the HTTP

VERBS easily and check your form-urlencoded, form-data, json, xml, text, and raw data.

SHARES

Normal Basic Auth Digest Auth OAuth 1.0 👁 No environment 🔧 ✖ 0

localhost:3000/todos POST 🔗 URL params 📄 Headers (0)

form-data x-www-form-urlencoded raw

name	Use Postman	✖
completed	true	✖
notes	Very cool	✖
Key	Value	

Send Preview Add to collection Reset

Body Headers (5) STATUS 200 OK TIME 26 ms

Pretty Raw Preview 📄 🔗 JSON XML

```
1 {  
2   "_v": 0,  
3   "name": "Use Postman",  
4   "completed": true,  
5   "_id": "542d7e97f3abec6c36903702"  
6 }
```

“Don't forget to check `x-www-form-urlencoded` or it won't work ;)

6.3 Websites and Mobile Apps

Probably these are the main consumers of APIs. You can interact with RESTful APIs using jQuery's `$ajax` and its wrappers, BackboneJS's Collections/models, AngularJS's `$http` or `$resource`, among many other libraries/frameworks and mobile clients.

In the end, we are going to explain how to use AngularJS to interact with this API.

7. ExpressJS Routes

To sum up we want to achieve the following:

Resource (URI)	POST (create)	GET (read)	PUT (update)	DELETE (destroy)
/todos	create new task	list tasks	error	error
/todos/:id	error	show task :id	update task :id	destroy task ID 1

Let's setup the routes

Create a new route called `todos.js` in the `routes` folder or rename `users.js`

```
1 mv routes/users.js routes/todos.js
```

In `app.js` add new `todos` route, or just replace `./routes/users` for `./routes/todos`

Adding todos routes

```
1 var todos = require('./routes/todos');
2 app.use('/todos', todos);
```

All set! Now, let's go back and edit our `routes/todos.js`.

7.1 List: GET /todos

Remember [mongoose query api](#)? Here's how to use it in this context:

`routes/todos.js`

```
1 var express = require('express');
2 var router = express.Router();
3
4 var mongoose = require('mongoose');
5 var Todo = require('../models/Todo.js');
6
```

SHARES

```
9   Todo.find(function (err, todos) {
10     if (err) return next(err);
11     res.json(todos);
12   });
13 });
14
15 module.exports = router;
```

Harvest time! We don't have any task in database but at least we verify it is working:

Testing all together

```
1  # Start database
2  mongod
3
4  # Start Webserver (in other terminal tab)
5  npm start
6
7  # Test API (in other terminal tab)
8  curl localhost:3000/todos
9  # => []%
```

diff

If it returns an empty array `[]` you are all set. If you get errors, try going back and making sure you didn't forget anything, or you can comment at the end of the post for help.

7.2 Create: POST /todos

Back in `routes/todos.js`, we are going to add the ability to create using [mongoose create](#). Can you make it work before looking at the next example?

routes/todos.js (showing just create route)

```
1
2  /* POST /todos */
3  router.post('/todos', function(req, res, next) {
```

SHARES

```
5     if (err) return next(err);
6     res.json(post);
7   });
8 });
```

diff

A few things:

- We are using the `router.post` instead of `router.get`.
- You have to stop and run the server again: `npm start`.

Everytime you change a file you have to stop and start the web server. Let's fix that using `nodemon` to refresh automatically:

Nodemon

```
1 # install nodemon globally
2 npm install nodemon -g
3
4 # Run web server with nodemon
5 nodemon
```

7.3 Show: GET /todos/:id

This is a snap with `Todo.findById` and `req.params`. Notice that `params` matches the placeholder name we set while defining the route. `:id` in this case.

routes/todos.js (showing just show route)

```
1 /* GET /todos/id */
2 router.get('/:id', function(req, res, next) {
3   Todo.findById(req.params.id, function (err, post) {
4     if (err) return next(err);
5     res.json(post);
6   });
7 });
```

SHARES

diff

Let's test what we have so far!

Testing the API with Curl

```
1 # Start Web Server on port 4000 (default is 3000)
2 PORT=4000 nodemon
3
4 # Create a todo using the API
5 curl -XPOST http://localhost:4000/todos -d 'name=Master%20Routes&cc
6 # => {"__v":0,"name":"Master Routes","completed":false,"note":"soon
7
8 # Get todo by ID (use the _id from the previous request, in my case
9 curl -XGET http://localhost:4000/todos/57a655997d2241695585ecf8
10 {"_id":"57a655997d2241695585ecf8","name":"Master Routes","completec
11
12 # Get all elements (notice it is an array)
13 curl -XGET http://localhost:4000/todos
14 [{"_id":"57a655997d2241695585ecf8","name":"Master Routes","complete
```

7.4 Update: PUT /todos/:id

Back in `routes/todos.js`, we are going to update tasks. This one you can do without looking at the example below, review [findByIdAndUpdate](#) and give it a try!

`routes/todos.js` (showing just update route)

```
1 /* PUT /todos/:id */
2 router.put('/:id', function(req, res, next) {
3   Todo.findByIdAndUpdate(req.params.id, req.body, function (err, post) {
4     if (err) return next(err);
5     res.json(post);
6   });
7 });
```

SHARES

curl update

```
1 # Use the ID from the todo, in my case 57a655997d2241695585ecf8
2 curl -XPUT http://localhost:4000/todos/57a655997d2241695585ecf8 -d '
3 # => {"_id":"57a655997d2241695585ecf8","name":"Master Routes","comp]
```

7.5 Destroy: DELETE /todos/:id

Finally, the last one! Almost identical to `update`, use `findByIdAndRemove`.

routes/todos.js (showing just update route)

```
1 /* DELETE /todos/:id */
2 router.delete('/:id', function(req, res, next) {
3   Todo.findByIdAndRemove(req.params.id, req.body, function (err, post) {
4     if (err) return next(err);
5     res.json(post);
6   });
7 });
```

diff

Is it working? Cool, you are done then! Is NOT working? take a look at the [full repository](#).

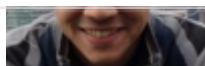
8. What's next?

Connecting AngularJS with this endpoint. Check out the [third](#) tutorial in this series.



Adrian Mejia is a full stack web developer working at Cigna in Boston

SHARES



writing books and posts about programming, technologies and nerdy stuff. Find our more [here](#).

Subscribe & stay up to date!

Subscribe

Tutorial MEAN Stack Series



**AngularJS Tutorial
for Beginners**

AngularJS tutorial for
beginners with NodeJS
ExpressJS and
MongoDB (Part I)

SHARES

Creating RESTful
APIs with



Creating RESTful APIs with NodeJS and MongoDB Tutorial (Part II)



MEAN Tutorial

MEAN Stack Tutorial MongoDB ExpressJS AngularJS NodeJS (Part III)

SHARES

[↑ Back to top](#) [✎ Edit](#)

Contents

1. What is a RESTful API?
2. Installing the MEAN Stack Backend
3. Using MongoDB with Mongoose
4. ExpressJS and Middlewares
5. Wiring up the MEAN Stack
6. API clients (Browser, Postman and curl)
7. ExpressJS Routes
8. What's next?

◀ NEWER

[MEAN Stack Tutorial MongoDB ExpressJS
AngularJS NodeJS \(Part III\)](#)

OLDER ▶

[AngularJS tutorial for beginners with NodeJS
ExpressJS and MongoDB \(Part I\)](#)

75 Comments

Adrian Mejia's Blog

[1 Login](#) ▼[♥ Recommend](#) 5[↗ Share](#)[Sort by Best](#) ▼

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)**Adam Carr** • 2 years ago

For anyone completing this tutorial using a current mongoose, I think the callback used in the `Todo.update` code sample is deprecated. Rather than getting `(err, numAffected, raw)` you now get `(err, result)` where `result` is an object with `"ok"`, `"nModified"`, and `"n"` keys. `result.nModified` will give you the number of rows affected. Hope that saves some time, cheers!

31 ^ | v • Reply • Share ›

**rjmil** ➔ Adam Carr • 2 years ago

SHARES



adriansky Mod → Adam Carr • 2 years ago

Yes, that's true. I just fixed it and updated the tutorials with latest mongoose, node and express

^ | v • Reply • Share ›



Abhijith S • 3 years ago

Its good and its helped me a lot.Thank you.

31 ^ | v • Reply • Share ›



David • 3 years ago

Awesome Tutorial. I could easily make my first steps with the MEAN stack and REST!
Thank you so much Adrian!!

With best regards from Germany,

David

31 ^ | v • Reply • Share ›



adriansky Mod → David • 2 years ago

You are welcome! Glad you liked it

^ | v • Reply • Share ›



Nathan Torquato • a year ago

Is there any live demo?

I've cloned the repo but it's not running locally. :/

30 ^ | v • Reply • Share ›



Ahmed • 3 years ago

Hi,

I think you forgot to include the step of installing mongoose. I had to 'npm install mongoose' before I was able to require('mongoose').

Great tutorial!

34 ^ | v • Reply • Share ›



adriansky Mod → Ahmed • 2 years ago

Thanks, I just updated the tutorial!

^ | v • Reply • Share ›



Luiz Guilherme Silva Junior • a year ago

Nice Post!

It helped me a lot while I was making a module that generate the endpoints based on your sequelize models.

SHARES

0 ^ | v • Reply • Share ›



Jonas Östlund • 3 years ago

As a clojure/lisp programmer, I appreciate your bottom-up approach to teaching. Start with the smallest building blocks and then assemble them into something meaningful.

5 ^ | v • Reply • Share ›



adriansky Mod → Jonas Östlund • 2 years ago

Glad you liked it!

^ | v • Reply • Share ›



hakeem ahmad • 3 years ago

why did you commit the "node_modules" directory? i think it should be ignored by git

2 ^ | v • Reply • Share ›



adriansky Mod → hakeem ahmad • 2 years ago

.gitignore was not set correctly. It's fixed now

^ | v • Reply • Share ›



Martin Jul Hammer • a year ago

Great article. Got my express js server up and running :)

1 ^ | v • Reply • Share ›



adriansky Mod → Martin Jul Hammer • a year ago

I'm glad to hear it!

1 ^ | v • Reply • Share ›



Ivo de Geus • a year ago

Thank you Adrian, this finally kickstarted me in MEAN development!

1 ^ | v • Reply • Share ›



adriansky Mod → Ivo de Geus • a year ago

Awesome! You are welcome!

^ | v • Reply • Share ›



BrahBassim • 3 years ago

Thank you very much for this tuts. The best i found.....

1 ^ | v • Reply • Share ›



Raul Abreu Leite • 3 years ago

I think this line is useless:

```
var mongoose = require('mongoose');
```

SHARES

Muito bom, tutorial! Obrigado!

1 ^ | v • Reply • Share ›



adriansky Mod ➔ Raul Abreu Leite • 2 years ago

Thanks! Removed!

^ | v • Reply • Share ›



Sai • 3 years ago

This was really helpful getting started with MEAN and REST!

1 ^ | v • Reply • Share ›



Lemaire Lucas • 10 months ago

Great tutorial !

^ | v • Reply • Share ›



William Romano • 10 months ago

Hi,

Great tutorial, I have an issue with it tho: On safari I only get the first value on my ng repeat, (if I send a static array I made in the scope it works) any ideas?

^ | v • Reply • Share ›



Евгений Бегунов • a year ago

Hi! Thank You for tutorial!

^ | v • Reply • Share ›



Anony User • a year ago

I was getting this error

MongoDB not working. "ERROR: dbpath (/data/db) does not exist."

Here is the solution for the same problem if any one is getting the same.

in terminal type

```
sudo mkdir -p /data/db/
```

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

```
monngod
```

it should start running...

thanks for the nice description, covers all basic steps.

^ | v • Reply • Share ›



Russ • a year ago

SHARES

^ | v • Reply • Share ›



Tim Owings • a year ago

This is a great tutorial even if I came to it late...couple of questions though. what is the reason you first create a project using npm init in the project directory and then create the main part of the app using express in the todo-api directory? Also, what is the reason for using node <cli> versus just editing a file...I apologize if these are stupid questions....Thanks!

^ | v • Reply • Share ›



Bendik B • a year ago

Hi! Great tutorial, I have one problem I really cannot find an answer to by Google. I successfully retrieve an empty array when using curl -XGET, but when i try to XPOST, I get a validation error. I am using curl -XPOST https://myurl:8080/todos -d 'name=MasterRoutes&completed=false-e=soon...'. I am posting the stacktrace below. I get the same error when using postman.

My Todo.js model is identical with yours.

I had to make one change to the routing, because express.Router() gave me an error when trying to call router.get, saying router was undefined. So i simply did router = express() and then router.get.

Any help greatly appreciated. Thanks!

Stacktrace:

```
ValidationError: Todo validation failed
at MongooseError.ValidationError (/home/ubuntu/workspace/todo-
api/node_modules/mongoose/lib/error/validation.js:23:11)
at model.Document.invalidate (/home/ubuntu/workspace/todo-
api/node_modules/mongoose/lib/document.js:1486:32)
at model.Document.set (/home/ubuntu/workspace/todo-
api/node_modules/mongoose/lib/document.js:753:10)
```

[see more](#)

^ | v • Reply • Share ›



Asuka Ishiyama • a year ago

this shit sucks...none of these work like it should and the walkthroughs are completely unclear

^ | v • Reply • Share ›



adriansky Mod ➔ Asuka Ishiyama • a year ago

Sorry to hear that. Where did you get lost?

^ | v • Reply • Share ›



Asuka Ishiyama ➔ adriansky • a year ago

down through step 3 with Mongoose and CRUD. Whenever i go to input the code, i keep getting errors and i cant go beyond that

SHARES



Yeah, I know it can be frustrating when things are not workig. If you post here the errors you are having I could give some pointers.

^ | v • Reply • Share ›



Dennis M Dewey • a year ago

Thank you so much for this well-structured tutorial on setting up an API using MongoDB and NodeJS ExpressJS.

I've been developing an Electron(atom-shell) application using AngularJS 1.x and my employers want me to connect it to an ASP.NET API that is using a MS SQL backend (I knowWTF?!?!?!?). Unfortunately I'm getting NTLM Authentication issues and cannot even use an API checker like POSTMAN to test/troubleshoot it. I'm going to try to sell them on using the appropriate technologies for this sort of project and Electron was not necessarily designed to work well with ASP.NET in the first place.

^ | v • Reply • Share ›



sreng khorn • a year ago

where is part I??

^ | v • Reply • Share ›



adriansky Mod ➔ sreng khorn • a year ago

It's here: <http://adrianmejia.com/blog...>

^ | v • Reply • Share ›



An Karthik • 2 years ago

Very nice and descriptive blog. Helped a lot.

Though I got stuck up at one point.

When we are trying CRUD using the Node CLI:

The following line is mentioned in the post:

"We can play with Mongoose in the console. In the todoApp type node to enter in the node CLI. Then:"

I am getting an error in the following command:

```
var mongoose = require('mongoose');
```

Cannot find module Mongoose.

Can anyone help me here.

^ | v • Reply • Share ›



adriansky Mod ➔ An Karthik • 2 years ago

You need to install it first "npm install mongoose" and the "node" and it would work

SHARES



mccat • 2 years ago

good job. i used this tut to build a different api and it worked. usually, even if you follow every step, something goes wrong because things were left out.

^ | v • Reply • Share ›



JL • 3 years ago

hi there. i've just attempted this on an ubuntu server but i'm not getting any results from my database. I had an existing mongo database so I decided to try to make a CRUD server using it. Please see my post here for details: <http://stackoverflow.com/qu...>

^ | v • Reply • Share ›



JL → JL • 3 years ago

I've resolve the issue. Mongoose / mongodb (?) was pluralizing my collection name and therefore the queries were failing. I had to pass the third argument to the mongoose.model() method and explicitly tell it what the name of the collection is. Voila. Everything is working.

1 ^ | v • Reply • Share ›



adriansky Mod → JL • 2 years ago

Great!

^ | v • Reply • Share ›



Simon Dupon • 3 years ago

O, I got it: when making post in POSTMAN you should select 'x-www-form-urlencoded' tab.

^ | v • Reply • Share ›



adriansky Mod → Simon Dupon • 2 years ago

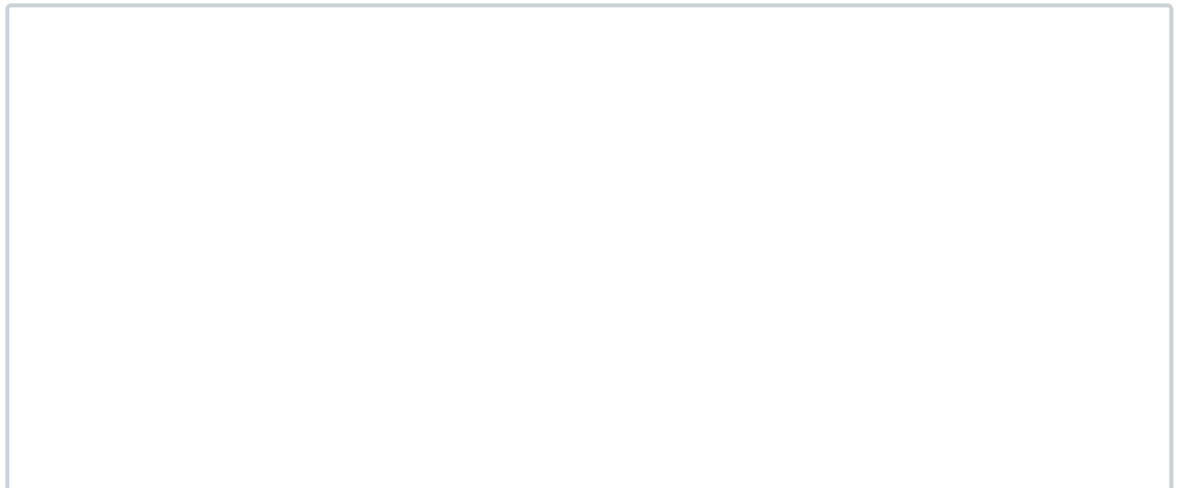
Yes, without that doesn't work

^ | v • Reply • Share ›



jainam shah → adriansky • a month ago

Hi even if it is checked I cannot make a POST call . It returns a Not Found 404 error . Can you please help . I'm stuck here All files are also identical



SHARES

[see more](#)[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Simon Dupon** • 3 years ago

My code is identical to yours, but I do not get 'name', 'completed' and 'note' data in responses, only '_id', '__v' and 'updated_at':

```
[  
  
{  
  
  "_id": "55a7ad265d8b9298144a75b0",  
  
  "__v": 0,  
  
  "updated_at": "2015-07-16T13:54:34.220Z"  
  
},  
  
{  
  
  "_id": "55a7ad3a5d8b9298144a75b1",  
  
  "__v": 0,  
  
  "updated_at": "2015-07-16T13:54:34.220Z"  
  
},  
  
]
```

Why?

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**adriansky** Mod ➔ Simon Dupon • 2 years ago

Also if you are using postman, make sure you have x-www-form-urlencoded selected

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**adriansky** Mod ➔ Simon Dupon • 2 years ago

Make sure todo.js is the same as the one in the project. Compare the final results with your

SHARES



johanness vix • 3 years ago

its a little verbose from my point of view. for a crud thats too much code.
any alternative which can be compared to a Spring Rests repository for Node?

^ | v • Reply • Share ›



Dimitri Mikadze • 3 years ago

Thank you!

© 2017 Adrian Mejia v.oxxds7

SHARES