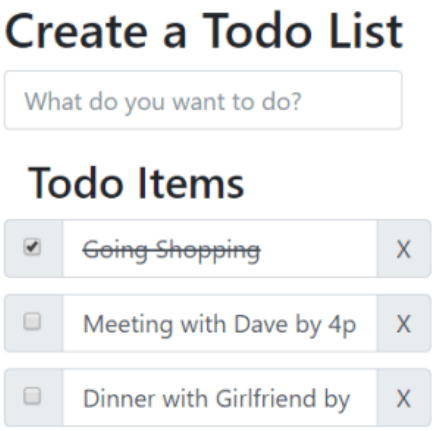
search

Build a Todo App with Node.Js, ExpressJs, MongoDB and VueJs – Part 1

Samuel James Feb 8 *Updated on Jul 15, 2018*

#vue #node **#expressjs**



In this tutorial, we will build a famous todo app with Node.Js using ExpressJs framework and MongoDB. Did I forget to tell you? The app will be API centric :).

The explosion of IoT has made developers start thinking API first, and before you wonder why I didn't title this post " **Build a Todo App API with Node.Js ExpressJs, MongoDB, and VueJs**"; I gotta tell ya, we'll build the client in part two of this post using Vue.Js. In a nutshell, if you want to learn how to build API with Node.Js, grab a bottle of beer, this post is for you.

What is ExpressJs

ExpressJs simply put: is a web framework for Node.Js - *stolen from the official docs* . Taylor Otwell (Laravel's creator) once said, "Developers build tools for developers". ExpressJs was built for developers with the goal of simplifying Node APIs.

Project Structure

You are probably going to get a shock of your life when I tell ya we need not more than 4 files with relatively few lines of code to build this backend if you are new to Node.Js.
To keep things simple, we'll create just two folders.

1. *app folder:* This houses our models, routes and the configuration file for the application.
2. *public* folder: This will contain our dear public files like an index.html page, images, CSS etc

This structure is not the best for large projects and as you may have thought, you would run into maintainability issues on larger projects.

I will be sharing my opinion on structuring medium to large Node.Js projects if you would remain a darling and check back in two weeks for the post.

What you will need to install

1. You will need Node

2. You will need to install MongoDB

Application Packages

You should create a project folder at this point to house all the source code. I'll call mine todo-app. This app depends on a couple of packages and will use npm to install them. Navigate to the project directory you just created and create a *package.* json file with the content below.

```
{
    "name": "node-todo",
    "version": "0.0.0",
    "description": "A simple todo application.",
    "main": "server.js",
    "author": "Samuel James",
    "dependencies": {
        "body-parser": "^1.5.2",
        "express": "~4.7.2",
        "method-override": "~2.1.2",
        "mongoose": "~3.6.2",
        "morgan": "^1.9.0"

    }
}
```

Run `npm install` to install the dependencies.

Landing Page

We create a landing page in the public folder with this:

```
<!--//path/to/poject/public/index.html-->

!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>VueJS NodeJS and Express  Todo App</title>

  </head>
  <body>
    <h1>Todo App</h1>
  </body>
</html>
```

App configuration file

Ofcourse, we'll need to define some configurations like database connection parameters and what port the application should run on.

/path/to/project/app/config.js file with the following content

```
module.exports = {
  DB: 'mongodb://localhost:27017/todos',
  APP_PORT: 4000
}
```

Todo Model

Model is an object representation of data in the database. So, we create a file at */path/to/project/app/Todo.Js* with the following:

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

// Define collection and schema for todo item

var todo = new mongoose.Schema({
  name: {
    type: String
  },

  done: {
    type: Boolean
  }
},

  {
    collection: 'todos'
  }
)

module.exports = mongoose.model('Todo', todo)
```

You notice we use mongoose for schema definition, right? Mongoose is an official MongoDB library built for manipulating MongoDB databases in Node.
name: A name field for todo item
done: Todo item status which is boolean

Routes

Every application has at least an entry point. A route in a web app is more like saying: "Hey Jackson when I ask ya for this,

give me that". Same goes for our app, we'll define what URL users need to access to get certain results or trigger certain actions.

In this case, we'll want users to be able to perform CRUD operations on todo items. We create a route file at */path/to/project/app/Routes.js* and update it with the following content:

```
'use strict'

var express = require('express')

var todoRoutes = express.Router()

var Todo = require('./Todo')

// get all todos in the db

todoRoutes.route('/all').get(function (req, res, next) {
  Todo.find(function (err, todos) {
    if (err) {
      return next(new Error(err))
    }

    res.json(todos) // return all todos
  })
})

// create a todo item
todoRoutes.route('/add').post(function (req, res) {
  Todo.create(
    {
      name: req.body.name,
      done: false
    },
    function (error, todo) {
```

```
      if (error) {
        res.status(400).send('Unable to create todo list')
      }
      res.status(200).json(todo)
    }
  )
})

// delete a todo item

todoRoutes.route('/delete/:id').get(function (req, res, next) {
  var id = req.params.id
  Todo.findByIdAndRemove(id, function (err, todo) {
    if (err) {
      return next(new Error('Todo was not found'))
    }
    res.json('Successfully removed')
  })
})

// perform update on todo item

todoRoutes.route('/update/:id').post(function (req, res, next) {
  var id = req.params.id
  Todo.findById(id, function (error, todo) {
    if (error) {
      return next(new Error('Todo was not found'))
    } else {
      todo.name = req.body.name
      todo.done = req.body.done

      todo.save({
        function (error, todo) {
          if (error) {
            res.status(400).send('Unable to update todo')
          } else {
            res.status(200).json(todo)
          }
        }
      })
    }
  })
```

```
})

module.exports = todoRoutes
```

First, you would want users to get a list of all to-do items existing in the database, hence we defined a route (*/all*) that accepts a get request and returns a JSON object of todo items if successful.

Our users like to get items as well as store new items, we added a route to create new to-do items. It accepts a post request. When Mr. A makes a post request to route (*/add*), a new to-do item is created in the database.

Once a todo-item is completed, we also want users to be able to mark it as done. To do this, one must know what item a user intends to mark as done in the first place. So, we defined an 'update route' with a route parameter which is the ID of the item to update.

Server File

Having defined all routes that our application needs, it is time to create an entry file which is the main file to run our project.

At the project root folder, create a *server.js* file and update it with this:

```
//server.js

/* jslint node: true */
'use strict'

var express = require('express')

var morgan = require('morgan')

var path = require('path')

var app = express()

var mongoose = require('mongoose')

var bodyParser = require('body-parser')

// Require configuration file defined in app/Config.js
var config = require('./app/Config')

// Connect to database
mongoose.connect(config.DB)

// Sends static files  from the public path directory
app.use(express.static(path.join(__dirname, '/public')))

// Use morgan to log request in dev mode
app.use(morgan('dev'))

app.use(bodyParser.json())

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

var port = config.APP_PORT || 4000

app.listen(port) // Listen on port defined in config file

console.log('App listening on port ' + port)

var todoRoutes = require('./app/Routes')
```

```
//  Use routes defined in Route.js and prefix it with api
app.use('/api', todoRoutes)

app.use(function (req, res, next) {
    // Website you wish to allow to connect
  res.setHeader('Access-Control-Allow-Origin', 'http://localhost:' + port)

    // Request methods you wish to allow
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH

    // Request headers you wish to allow
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type

    // Pass to next layer of middleware
  next()
})
// Server index.html page when request to the root is made
app.get('/', function (req, res, next) {
  res.sendfile('./public/index.html')
})
```

We required:

- Express Js
- morgan - *a logging middleware for Node*
- path - a *package for working with file and directory*
- mongoose - *a package for working with MongoDB*
- body-parser - *a body parsing middleware*

We set our app to listen on the port set in *app/Config.js*. if it's not set, it defaults to 4000. We also tell it to use routes defined in *app/Routes.js* and prefix with api.

To start the application, navigate to project root where server.js file is located and run `node server.js`

Let's create a new todo item

```
$ curl -H "Content-Type: application/json" -X POST -d "{\"name\":\"Going Shopp

{"__v":0,"name":"Going Shopping","done":false,"_id":"5a6365a39a2e56bc54000003"
```

Get all todo items

```
$ curl  http://localhost:4000/api/all

[{"_id":"5a6365a39a2e56bc54000003","name":"Doing Laundry","done":false,"__v":0
```

Get source code

Read part 2 now

Samuel James + FOLLOW

Gym,code and everything in between.

samuelabiodunj abiodunjames samuelabiodun.com

Add to the discussion

PREVIEW SUBMIT

Pek Bun Oh

Mar 8

Hi Samuel,thanks for the tutorial!

I clone from your repo and run on my mac with node v8.9.4.

After run node server.js, 127.0.0.1:4000 works fine. But running curl command like your:

$ curl -H "Content-Type: application/json" -X POST -d '{"name":"Going Shopping"}' localhost:4000/api/add

the server just hanging there not responding anything:

Note: Unnecessary use of -X or --request, POST is already inferred.

- Trying 127.0.0.1...
- TCP_NODELAY set
- Connected to 127.0.0.1 (127.0.0.1) port 4000 (#0) > POST /api/add HTTP/1.1 > Host: 127.0.0.1:4000 > User-Agent: curl/7.54.0 > Accept: / > Content-Type: application/json > Content-Length: 25 >
- upload completely sent off: 25 out of 25 bytes

After Ctrl+C, server showed:
node server.js
App listening on port 4000
POST /api/add - - ms - -

Could you help me out here? Thank you.

2 REPLY

Akash Jobanputra

Apr 7

I was facing the same issue, after hours of searching, I found that downgrading the version of mongoose to 4.7.6 works.
I used below command to downgrade the mongoose module, and then the API worked perfectly.

```
npm install mongoose@4.7.6
```

3

REPLY

Zsofia Boldizsar

Jun 25

Thanks for sharing this, Akash. I should have looked at the comments section in the first place. I had the same issue, but with your tip, I managed to solve it.

1

REPLY

Triệu Vĩnh Viêm

Jun 11

We can use Postman app to send request form get/post. You will receive response data.

1

REPLY

Samuel James

Apr 19

Hi Pek,

My bad! I'm just seeing this. I apologize for my late response.
Hope downgrading solves your problem as suggested by @Akash Jobanputra

REPLY

Sam Benskin

Feb 12

Good article, small suggestion if I may. The use of third party libraries like Morgan and so on is good but it would have been better to introduce them later on and just keep it super simple to begin with. Then as we're reading we can see the code build up from the simple to the more complex as you add in each part.

Other than that, a great read, thanks

2

REPLY

Samuel James

Feb 12

Oh my Gosh, I overlooked that.

Thanks, Sam. I'll put that into consideration in my next post.

10

REPLY

Natalia Vidal

Jul 12

Also the first error I got was that morgan could not be found..I've installed it and it's all working now but it isn't on the package.json nor does it say to install it anywhere.

1

THREAD

Samuel James

Jul 15

Sorry @Natalia, The article has been updated to include Morgan.

1

REPLY

unika

Jun 21

Hi Samuel!

I followed your first part of the tutorial, but my curl request didn't work.
And my app didn't work, but my IDE didn't produce me any mistakes...

UPDATE

You missed the code:

```
app.listen(port, (err) => {
if (err) {
return console.log('something bad happened', err)
}
console.log( `server is listening on ${port}` )
})
```

My app works with it.

UPDATE

This: curl -H "Content-Type: application/json" -X POST -d '{"name":"Going Shopping"}' localhost:4000/api/add

Show me this:

Unexpected token ' in JSON at position 0

Please, could you help me with it?

1 REPLY

Samuel James

Jun 22

Hello Unika,

Sorry for my late response. Use this:

```
curl -X POST -H "Content-Type: application/json" \
-d "{\"name\":\"Going shopping\"}" http://localhost:4000/api/add
```

2 REPLY

Ben Halpern

Hey there, we see you aren't signed in. (Yes you, the reader. This is a fake comment.)

Please consider creating an account on dev.to. It literally takes a few seconds and ***we'd appreciate the support so much***.

Plus, no fake comments when you're signed in.

JOIN THE DEV COMMUNITY

Thoby V ijishakin

Feb 8

Nice article, so awesome and simple to follow

1 REPLY

Samuel James

Feb 8

Thanks for reading, Thoby

2

Thoby V ijishakin

REPLY

Feb 9

Welcome Sam! Was enlightening

1

REPLY

Mark Johnson

Apr 3

This is really awesome, thanks for putting it together! On to part 2 :) Maybe you could edit this post with a link to the second part?

2

REPLY

Samuel James

May 6

Thank you for this. It's now updated with the link to Part 2

1

REPLY

code of conduct - report abuse

Classic DEV Post from Oct 19 '17

19 Types of Developers Explained

Lorenzo Pasqualis

In this post, I define 19 of the most common types of developers with a short description and list of technologies they use and skills they must have.

170 31

READ POST SAVE FOR LATER

Another Post You Might Like

Protect your Node.js app from Cross-Site Request Forgery

Dominik Kundel

Cross Site Request Forgery aka CSRF/XSRF is used by attackers to perform requests on behalf of others. Learn how to protect your Node.js app from it.

114

READ POST SAVE FOR LATER

You don't need express to get started with socket.io

Sadick - Aug 8

100 Days of Code Challenge

Devin W. Leaman - Aug 8

Adventures of a Hobbyist ~ Part One

Andrew Bone - Aug 7

How to Create Your Own Cryptocurrency Blockchain in Node.js

Dr. Michael Garbade - Aug 7
