

Creating a Single Page Todo App with Node and Angular

Chris Sevilleja (<http://scotch.io/author/chris>) November 7, 2013
angularJS (<http://scotch.io/tag/angular-js>), MEAN (<http://scotch.io/tag/mean>), node.js (<http://scotch.io/tag/node-js>)
120 Comments (http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular#disqus_thread)

352



Share

243

(<http://scotch.io/popular>)

Our Most Popular Articles
(<http://scotch.io/popular>)



Tweet ([https://twitter.com/intent/tweet?](https://twitter.com/intent/tweet?text=Creating+a+Single+Page+Todo+App+with+Node+and+Angular&url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular&via=scotch_io&related=s)

text=Creating+a+Single+Page+Todo+App+with+Node+and+Angular&url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular&via=scotch_io&related=s

225



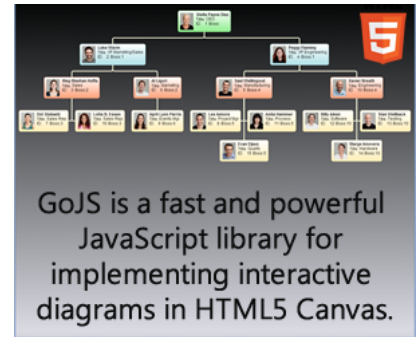
Plus ([https://plus.google.com/share?](https://plus.google.com/share?url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular)

url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular

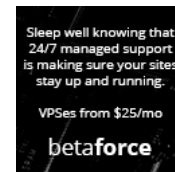
</> View Code (<https://github.com/scotch-io/node-todo/tree/tut1-starter>)

View Demo (<http://scotch-node-todo.azurewebsites.net/>)

Advertisements (<http://scotch.io/advertise>)



(http://stats.buysellads.com/click.go?z=1291897&b=4691598&g=&s=&sw=1360&sl=1360&utm_source=scotchio&utm_medium=tr300x250colorful&utm_campaign=bsa-gojs)



(http://stats.buysellads.com/click.go?z=1291908&b=4794368&g=&s=&sw=1360&sl=1360&utm_source=scotchio&utm_medium=web)

Advertise Here



Best of Sublime Text

(<http://scotch.io/bar-talk/best-of-sublime-text-3-features-plugins-and-settings>)

Best of Sublime Text 3: Features, Plugins, and Settings (<http://scotch.io/bar-talk/best-of-sublime-text-3-features-plugins-and-settings>)



(<http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular>)

Creating a Single Page Todo App with Node and Angular
(<http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular>)

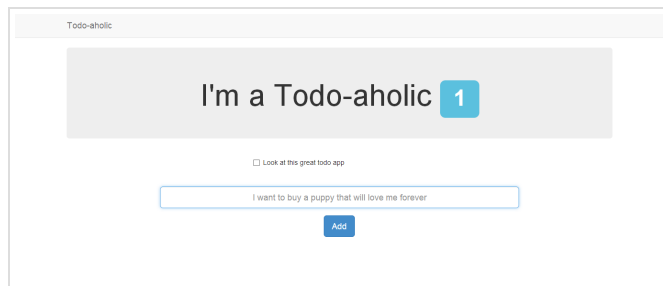
Single Page Application with Node and Angular

Today we will be creating a very simple Todo application using the MEAN (Mongo, Express, Angular, Node) stack. We will be creating:

- Single page application to create and finish todos
- Storing todos in a MongoDB using Mongoose
- Using the Express framework
- Creating a RESTful Node API
- Using Angular for the frontend and to access the API

While the application is simple and **beginner to intermediate** level in its own right, the concepts here can apply to much more advanced apps. The biggest things we should focus on is using Node as an API and Angular as the frontend. Making them work together can be a bit confusing so this tutorial should help alleviate some confusion. Buckle those seatbelts; this could be a long one.

What We'll Be Building



(<http://scotch.io/wp-content/uploads/2013/11/todoaholic.png>)

🔗 Base Setup

File Structure

We are going to keep the file structure very simple and put most of the code for our Node application into the `server.js` file. In larger applications, this should be broken down further to separate duties. Mean.io (<http://mean.io>) is a good boilerplate to see best practices and how to separate file structure. Let's go ahead and create our simpler file structure and edit the files as we go along.



(<http://scotch.io/tutorials/javascript/angularjs-form-validation>)

AngularJS Form Validation
(<http://scotch.io/tutorials/javascript/angularjs-form-validation>)



(<http://scotch.io/tutorials/javascript/single-page-apps-with-angularjs-routing-and-templating>)

Single Page Apps with AngularJS Routing and Templating
(<http://scotch.io/tutorials/javascript/single-page-apps-with-angularjs-routing-and-templating>)



(<http://scotch.io/tutorials/javascript/angular-routing-using-ui-router>)

AngularJS Routing Using UI-Router
(<http://scotch.io/tutorials/javascript/angular-routing-using-ui-router>)



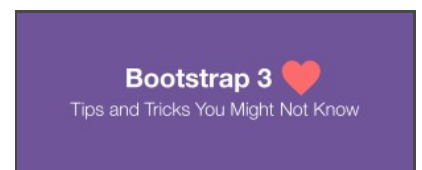
(<http://scotch.io/tutorials/javascript/easy-node-authentication-setup-and-local>)

Easy Node Authentication: Setup and Local
(<http://scotch.io/tutorials/javascript/easy-node-authentication-setup-and-local>)



(<http://scotch.io/tutorials/javascript/submitting-ajax-forms-the-angularjs-way>)

Submitting AJAX Forms: The AngularJS Way
(<http://scotch.io/tutorials/javascript/submitting-ajax-forms-the-angularjs-way>)



(<http://scotch.io/bar-talk/bootstrap-3-tips->)

```

- public          <!-- holds all our files for our front-end -->
- core.js         <!-- all angular code for our app -->
- index.html      <!-- main view -->
- package.json    <!-- npm configuration to install dependencies -->
- server.js       <!-- Node configuration -->

```

Installing Modules

In Node, the `package.json` file holds the configuration for our app. Node's package manager (npm) will use this to install any dependencies or modules that we are going to use. In our case, we will be using Express (<http://expressjs.com/>) (popular Node framework) and Mongoose (<http://mongoosejs.com/>) (object modeling for MongoDB).

```

// package.json
{
  "name"       : "node-todo",
  "version"    : "0.0.0",
  "description": "Simple todo application.",
  "main"       : "server.js",
  "author"     : "Scotch",
  "dependencies": {
    "express"   : "~3.4.4",
    "mongoose"  : "~3.6.2"
  }
}

```

Now if we run `npm install`, npm will look at this file and install Express and Mongoose.

```

$ npm install
npm http GET https://registry.npmjs.org/express
npm http GET https://registry.npmjs.org/mongoose
npm http 304 https://registry.npmjs.org/express
npm http 304 https://registry.npmjs.org/mongoose
npm http GET https://registry.npmjs.org/commander/1.3.2
npm http GET https://registry.npmjs.org/connect/2.11.0
npm http GET https://registry.npmjs.org/range-parser/0.0.4
npm http GET https://registry.npmjs.org/mkdirp/0.3.5
npm http GET https://registry.npmjs.org/cookie/0.1.0
npm http GET https://registry.npmjs.org/buffer-crc32/0.2.1
npm http GET https://registry.npmjs.org/fresh/0.2.0
npm http GET https://registry.npmjs.org/methods/0.1.0
npm http GET https://registry.npmjs.org/send/0.1.4
npm http GET https://registry.npmjs.org/cookie-signature/1.0.1
npm http GET https://registry.npmjs.org/debug
npm http GET https://registry.npmjs.org/hooks/0.2.1
npm http GET https://registry.npmjs.org/ms/0.1.0
npm http GET https://registry.npmjs.org/mongodb/1.3.19
npm http GET https://registry.npmjs.org/sliced/0.0.5
npm http GET https://registry.npmjs.org/muri/0.3.1
npm http GET https://registry.npmjs.org/mpromise/0.2.1
npm http GET https://registry.npmjs.org/npath/0.1.1
npm http GET https://registry.npmjs.org/regexp-clone/0.0.1
npm http 304 https://registry.npmjs.org/range-parser/0.0.4
npm http 304 https://registry.npmjs.org/commander/1.3.2
npm http 304 https://registry.npmjs.org/mkdirp/0.3.5
npm http 304 https://registry.npmjs.org/cookie/0.1.0
npm http 304 https://registry.npmjs.org/buffer-crc32/0.2.1

```

(<http://scotch.io/wp-content/uploads/2013/11/npm-install.png>)

Node Configuration

In our `package.json` file, we told it that our main file would be `server.js`. This is the main file for our Node app and where we will configure the entire application.

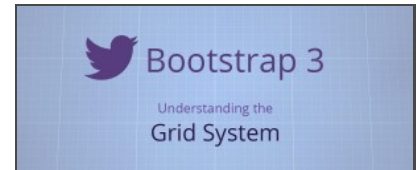
and-tricks-you-might-not-know)

Bootstrap 3 Tips and Tricks You Might Not Know (<http://scotch.io/bar-talk/bootstrap-3-tips-and-tricks-you-might-not-know>)



(<http://scotch.io/tutorials/simple-laravel-crud-with-resource-controllers>)

Simple Laravel CRUD with Resource Controllers (<http://scotch.io/tutorials/simple-laravel-crud-with-resource-controllers>)



(<http://scotch.io/bar-talk/understanding-the-bootstrap-3-grid-system>)

Understanding the Bootstrap 3 Grid System (<http://scotch.io/bar-talk/understanding-the-bootstrap-3-grid-system>)

This is the file where we will:

- Configure our application
- Connect to our database
- Create our Mongoose models
- Define routes for our RESTful API
- Define routes for our frontend Angular application
- Set the app to listen on a port so we can view it in our browser

For now, we will just configure the app for Express, our MongoDB database, and listening on a port.

```
// server.js

// set up =====
var express = require('express');
var app     = express();
var mongoose = require('mongoose');

// configuration =====

mongoose.connect('mongodb://node:node@mongo.onmodulus.net');

app.configure(function() {
  app.use(express.static(__dirname + '/public'));
  app.use(express.logger('dev'));
  app.use(express.bodyParser());
});

// listen (start app with node server.js) =====
app.listen(8080);
console.log("App listening on port 8080");
```

Just with that bit of code, we now have an HTTP server courtesy of Node. We have also created an app with Express and now have access to many benefits of it. In our `app.configure` section, we are using express modules to add more functionality to our application.

Database Setup

We will be using a remote database hosted on Modulus.io (<https://modulus.io/>). They provide a great service and give you \$15 upfront to use as you see fit. This is great for doing testing and creating databases on the fly.

Modulus will provide the database URL you need and you can use `mongoose.connect` to connect to it. That's it.

Start Your App!

Now that we have our `package.json` and `server.js` started up, we can start up our server and see what's going on. Just go into your console and use the following command:

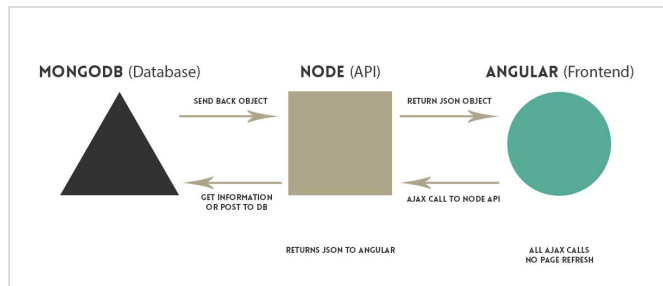
```
node server.js
```

Now you have a server listening on port 8080. You can't see anything in your browser at **`http://localhost:8080`** yet since we didn't configure our application to output anything. But it's a start!

Automatically restart server when files change: By default, node will not monitor for file changes after your server has been started. This means you'd have to shut down and start the server every time you made a file change. This can be fixed with **nodemon**. To use: install nodemon globally `npm install -g nodemon`. Start your server with `nodemon server.js` now. Smooth sailing from there.

🔗 Application Flow

Now a brief overview of how all our moving parts will work together. There are a lot of different ideas and technologies involved in this application that it is easy to get mixed up with them all. In our diagram below, we explain a bit of the separation of tasks and how the parts tie in together.



(<http://scotch.io/wp-content/uploads/2013/11/mean.jpg>)

Angular is on its own in the frontend. It accesses all the data it needs through the Node API. Node hits the database and returns JSON information to Angular based on the RESTful routing.

This way, you can separate the frontend application from the actual API. If you want to extend the API, you can always build more routes and functions into it without affecting the frontend Angular application. This way you can eventually build different apps on different platforms since you just have to hit the API.

🔗 Creating Our Node API

Before we get to the frontend application, we need to create our RESTful API. This will allow us to have an api that will **get all todos**, **create a todo**, and **complete and delete a todo**. It will return all this information in JSON format.

Todo Model

We must define our model for our Todos. We'll keep this simple. After the **configuration section** and before the **listen** section, we'll add our model.

```
// define model =====  
var Todo = mongoose.model('Todo', {  
  text : String  
});
```

That is all we want. Just the text for the todo. MongoDB will automatically generate an `_id` for each todo that we create also.

RESTful API Routes

Let's generate our Express routes to handle our API calls.

```

// server.js
...

// routes =====

// api -----
// get all todos
app.get('/api/todos', function(req, res) {

    // use mongoose to get all todos in the database
    Todo.find(function(err, todos) {

        // if there is an error retrieving, send the error
        if (err)
            res.send(err)

        res.json(todos); // return all todos in JSON format
    });
});

// create todo and send back all todos after creation
app.post('/api/todos', function(req, res) {

    // create a todo, information comes from AJAX request
    Todo.create({
        text : req.body.text,
        done : false
    }, function(err, todo) {
        if (err)
            res.send(err);

        // get and return all the todos after you create
        Todo.find(function(err, todos) {
            if (err)
                res.send(err)
            res.json(todos);
        });
    });
});

// delete a todo
app.delete('/api/todos/:todo_id', function(req, res) {
    Todo.remove({
        _id : req.params.todo_id
    }, function(err, todo) {
        if (err)
            res.send(err);

        // get and return all the todos after you create
        Todo.find(function(err, todos) {
            if (err)
                res.send(err)
            res.json(todos);
        });
    });
});
...

```

Based on these routes, we've built a table to explain how a frontend application should request data from the API.

HTTP Verb	URL	Description
GET	/api/todos	Get all of the todos
POST	/api/todos	Create a single todo

DELETE	/api/todos/:todo_id	Delete a single todo
--------	---------------------	----------------------

Inside of each of our API routes, we use the Mongoose actions to help us interact with our database. We created our Model earlier with `var Todo = mongoose.model` and now we can use that to **find**, **create**, and **remove**. There are many more things you can do and I would suggest looking at the official docs (<http://mongoosejs.com/docs/guide.html>) to learn more.

Our API is done! Rejoice! If you start up your application, you can interact with it at `localhost:8080/api/todos` to get all the todos. There won't be anything currently since you haven't added any.

🔗 Frontend Application with Angular

We have **created a Node application**, **configured our database**, **generated our API routes**, and **started a server**. So much already done and still a little bit longer to go!

The work that we've done so far can stand on its own as an application. It can be an API we use let applications and users connect with our content.

We want to be the first to use our brand new API that we've just created. This is one of my favorite terms that I learned about last month: We will be dogfooding (http://en.wikipedia.org/wiki/Eating_your_own_dog_food). We could treat this as we are our very first client to use our new API. We are going to keep this simple so we'll have just our `index.html` and `core.js` to define our frontend.

Defining Frontend Route

We have already defined our API routes. Our application's API is accessible from `/api/todos`, but what about our frontend? How do we display the `index.html` file at our home page?

We will add one route to our `server.js` file for the frontend application. This is all we need to do since Angular will be making a single page application and handle the routing.

After our API routes, and before the `app.listen`, add this route:


```
// server.js
...
// application -----
app.get('*', function(req, res) {
  res.sendFile('./public/index.html'); // load the single
});
...
```

This will load our single index.html file when we hit
localhost:8080.

Setting Up Angular core.js

Let's go through our Angular setup first. We have to **create a module**, **create a controller**, and **define functions to handle todos**. Then we can **apply to view**.

```
// public/core.js
var scotchTodo = angular.module('scotchTodo', []);

function mainController($scope, $http) {
  $scope.formData = {};

  // when landing on the page, get all todos and show them
  $http.get('/api/todos')
    .success(function(data) {
      $scope.todos = data;
      console.log(data);
    })
    .error(function(data) {
      console.log('Error: ' + data);
    });

  // when submitting the add form, send the text to the node
  $scope.createTodo = function() {
    $http.post('/api/todos', $scope.formData)
      .success(function(data) {
        $scope.formData = {}; // clear the form so our
        $scope.todos = data;
        console.log(data);
      })
      .error(function(data) {
        console.log('Error: ' + data);
      });
  };

  // delete a todo after checking it
  $scope.deleteTodo = function(id) {
    $http.delete('/api/todos/' + id)
      .success(function(data) {
        $scope.todos = data;
        console.log(data);
      })
      .error(function(data) {
        console.log('Error: ' + data);
      });
  };
}
```

We create our Angular module (`scotchApp`) and controller (`mainController`).

We also create our functions to **get all todos**, **create a todo**, and **delete a todo**. All these will be hitting the API we just created. On page load, we will `GET /api/todos` and bind the JSON we receive from the API to `$scope.todos`. We will then loop over these in our view to make our todos.

We'll follow a similar pattern for **creating and deleting**. Run our action, remake our todos list.

SCOTCH
([HTTP://SCOTCH.IO](http://scotch.io))



BAR TALK
([HTTP://SCOTCH.IO/BAR-TALK](http://scotch.io/bar-talk))



TUTORIALS
([HTTP://SCOTCH.IO/TUTORIALS](http://scotch.io/tutorials))



SERIES
([HTTP://SCOTCH.IO/SERIES](http://scotch.io/series))



QUICK TIPS
([HTTP://SCOTCH.IO/QUICK-TIPS](http://scotch.io/quick-tips))



PROJECTS
([HTTP://SCOTCH.IO/WORK](http://scotch.io/work))



ABOUT
([HTTP://SCOTCH.IO/ABOUT](http://scotch.io/about))

Frontend View `index.html`

Here we will keep it simple. This is the HTML needed to interact with Angular. We will:

- Assign Angular module and controller
- Initialize the page by getting all todos
- Loop over the todos
- Have a form to create todos
- Delete todos when they are checked



Get Our Latest Articles

Your Secret Electronic Address

Subscribe

```

<!-- index.html -->
<!doctype html>

<!-- ASSIGN OUR ANGULAR MODULE -->
<html ng-app="scotchTodo">
<head>
  <!-- META -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>Node/Angular Todo App</title>

  <!-- SCROLLS -->
  <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
  <style>
    html { overflow-y:scroll; }
    body { padding-top:50px; }
    #todo-list { margin-bottom:30px; }
  </style>

  <!-- SPELLS --&>
  <script src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
  <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.23/angular.min.js"></script>
  <script src="core.js"></script>

</head>
<!-- SET THE CONTROLLER AND GET ALL TODOS -->
<body ng-controller="mainController">
  <div class="container">

    <!-- HEADER AND TODO COUNT -->
    <div class="jumbotron text-center">
      <h1>I'm a Todo-aholic <span class="label label-info">{{ todos.length }}</span>
    </div>

    <!-- TODO LIST -->
    <div id="todo-list" class="row">
      <div class="col-sm-4 col-sm-offset-4">

        <!-- LOOP OVER THE TODOS IN $scope.todos -->
        <div class="checkbox" ng-repeat="todo in todos">
          <label>
            <input type="checkbox" ng-click="delete(todo)" /> {{ todo }}
          </label>
        </div>

      </div>
    </div>

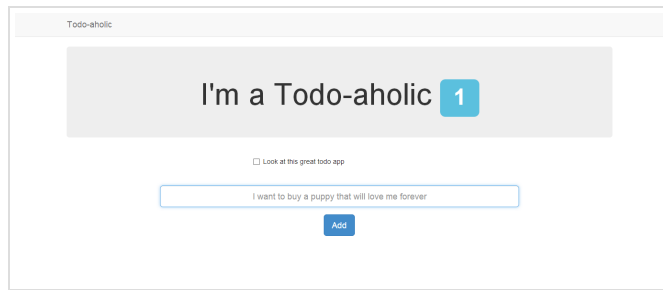
    <!-- FORM TO CREATE TODOS -->
    <div id="todo-form" class="row">
      <div class="col-sm-8 col-sm-offset-2 text-center">
        <form>
          <div class="form-group">
            <!-- BIND THIS VALUE TO formData.text -->
            <input type="text" class="form-control" value="{{ formData.text }}" />
          </div>
          <!-- createToDo() WILL CREATE NEW TODOS -->
          <button type="submit" class="btn btn-primary">Add</button>
        </form>
      </div>
    </div>

  </div>

</body>
</html>

```

Take a look at what we have.



(<http://scotch.io/wp-content/uploads/2013/11/todoaholic.png>)

Conclusion

Now we have a fully working application that will show, create, and delete todos all via API (that we built!). That was quite a day. We've done so much. Just an overview of what we've accomplished:

- RESTful Node API using Express
- MongoDB interaction using mongoose
- Angular AJAX \$http calls
- Single page application w/ no refreshes
- Dogfooding (sorry, I really like that word)

Test the Application

Go ahead and download the code on Github (<https://github.com/scotch-io/node-todo>) and tweak it or test it. To get it all up and running:

1. Make sure you have Node and npm (<http://nodejs.org/>) installed
2. Clone the repo:

```
git clone git@github.com:scotch-io/node-todo
```
3. Install the application: `npm install`
4. Start the server: `node server.js`
5. View in your browser at `http://localhost:8080`

I hope this was insightful on how to have lots of moving parts work together. In the future, we will look at separating our `server.js` file since that got a little crazy.

Further Reading

If you are interested in more MEAN stack applications, we've written up a guide to get you started in building your own MEAN stack foundation.

Setting Up a MEAN Stack Single Page Application (</bar-talk/setting-up-a-mean-stack-single-page-application>)

Edit #1: Removing ng-init

This article is part of our [Node and Angular To-Do App \(/series/node-and-angular-to-do-app\)](#) series.

1. Creating a Single Page To-do App with Node and Angular
2. [Node Application Organization and Structure \(/tutorials/javascript/node-and-angular-to-do-app-application-organization-and-structure\)](#)
3. [Angular Modules: Controllers and Services \(/tutorials/javascript/node-and-angular-to-do-app-controllers-and-services\)](#)

</> View Code (<https://github.com/scotch-io/node-todo/tree/tut1-starter>)

🌐 View Demo (<http://scotch-node-todo.azurewebsites.net/>)

352



Share

243

Tweet ([https://twitter.com/intent/tweet?](https://twitter.com/intent/tweet?text=Creating+a+Single+Page+Todo+App+with+Node+and+Angular&url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular&via=scotch_io&related=scotch_io)[text=Creating+a+Single+Page+Todo+App+with+Node+and+Angular&url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular&via=scotch_io&related=scotch_io](https://twitter.com/intent/tweet?text=Creating+a+Single+Page+Todo+App+with+Node+and+Angular&url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular&via=scotch_io&related=scotch_io))

225

Plus ([https://plus.google.com/share?](https://plus.google.com/share?url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular)[url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular](https://plus.google.com/share?url=http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular))

Related Articles



(<http://scotch.io/tutorials/javascript/node-and-angular-to-do-app-application-organization-and-structure>)

Node and Angular To-Do App: Application Organization and Structure
(<http://scotch.io/tutorials/javascript/node-and-angular-to-do-app-application-organization-and-structure>)



(<http://scotch.io/tutorials/javascript/node-and-angular-to-do-app-controllers-and-services>)

Node and Angular To-Do App: Controllers and Services
(<http://scotch.io/tutorials/javascript/node-and-angular-to-do-app-controllers-and-services>)



(<http://scotch.io/bar-talk/setting-up-a-mean-stack-single-page-application>)

Setting Up a MEAN Stack Single Page Application (<http://scotch.io/bar-talk/setting-up-a-mean-stack-single-page-application>)



(<http://scotch.io/tutorials/php/create-a-laravel-and-angular-single-page-comment-application>)

Create a Laravel and Angular Single Page Comment Application
(<http://scotch.io/tutorials/php/create-a-laravel-and-angular-single-page-comment-application>)



(<http://scotch.io/tutorials/javascript/single-page-apps-with-angularjs-routing-and-templating>)

Single Page Apps with AngularJS Routing and Templating

(<http://scotch.io/tutorials/javascript/single-page-apps-with-angularjs-routing-and-templating>)



(<http://scotch.io/tutorials/javascript/build-a-restful-api-using-node-and-express-4>)

Build a RESTful API Using Node and Express 4

(<http://scotch.io/tutorials/javascript/build-a-restful-api-using-node-and-express-4>)



Chris Sevilleja (<http://scotch.io/author/chris>)

Design, development, and anything in between that I find interesting.

♥ View My Contributions (<http://scotch.io/author/chris>)



(<https://plus.google.com/u/0/111211028394313645953>)

Stay Connected With Us
hover these for magic



SUBSCRIBE



FOLLOW



LIKE



+1



Get valuable tips, articles, and resources straight to your inbox. Every Tuesday.



120 Comments

Scotch

Login ▾

Sort by Best ▾

Share Favorite





Sly Cooper • 6 months ago

Those...social network...buttons...hnnng.

11 ^ | ▾ • Reply • Share ›



Shawn → Sly Cooper • 3 months ago

I wanted to say something about them too.. but hey.. they got our attention.

^ | ▾ • Reply • Share ›



user1 • 4 months ago

Resource interpreted as Script but transferred with MIME type text/html: "http://localhost:8080/ToDo/public/js/core.js".

and browser throws following errors in this example

Uncaught SyntaxError: Unexpected token < core.js:2

Uncaught Error: No module: scotchTodo angular.min.js:18

5 ^ | ▾ • Reply • Share ›



CodySCarlson → user1 • a month ago

@user1, Exactly (arrrrg!)... The reason for this is that when you ask the server for "core.js" (along with *anything else), the server sees a '/'... ('/core.js' for your request) and still serves the 'index.html' file. The promise in Express says "app.get('*', ...)" -- mind the *all character -- this is the same as "app.get('//*', ...)";. Meaning, ALL GET requests -- including requests of '/api/...' GETs (though route ordering may affect this). The only way I can see getting around this is to namespace your app.get to 'app.get('/namespace/*', ...)'; AND have all your Angular Routes begin with this: e.g.: 'when('/namespace/r/o/u/t/e', { ... })'.

I just can't see a way to only 'redirect all' for only those requests which are not for other assets (js, css, img, etc).

HAS ANYONE GOTTEN AROUND THIS IN AN ELEGANT, SCALABLE/LOOSELY-COUPLED WAY???

^ | v • Reply • Share ›



Eric Swann → CodySCarlson • 15 days ago

I wouldn't call it elegant, but I just put a route in front of the catchall to serve the script specifically. So when getting core.js, it hits that route first and stops. Ex:

```
app.get("/public/core.js", function (req, res) {
  res.sendFile("./public/core.js");
});
app.get("/*", function (req, res) {
  res.sendFile("./public/index.html");
});
```

Then I discovered that I just had the static file directory set incorrectly. When I fixed it, everything worked:

```
app.use(express.static(__dirname + 'public'));
```

^ | v • Reply • Share ›



Thomas Stock • 8 months ago

Thanks a lot, this was very educational and well written.

5 ^ | v • Reply • Share ›



Chris Sevilleja Bartender → Thomas Stock • 8 months ago

Thanks for reading!

1 ^ | v • Reply • Share ›



vjennings • 4 months ago

wonderful tutorial!! looking forward to more of these!

how is angular identifying the data coming from mongodb? where are we telling angular that each 'todo' corresponds to an instance in the 'Todo' model? does it bind to them by name (i.e. does it say that todo == Todo and binds accordingly?) suppose we had another model in the database called 'Modo,' would we have to call the data 'modo' in angular so that it knows to bind to it on the server-side?

thanks again for all your effort! this is one of the highest quality tutorials i've seen!

3 ^ | v • Reply • Share ›



Chris Sevilleja Bartender → vjennings • 4 months ago

Hello. Angular doesn't identify the data coming from mongo. Node does that. In our Node backend, when we defined our Mongoose model, Mongoose will automatically look for a todo in the database. We then return that data to Angular when Angular requests it using the \$http.get.

All Angular knows is the URL to use when getting data (/api/todos) and that it's being returned data.

2 ^ | v • Reply • Share ›



Jesus Rodriguez • 6 months ago

Hey Chris, awesome page, I love the design. I came here to read about node because I want to learn it. I checked this serie to see how you integrate Angular.js into Node.

I am an Angular.js developer/author and here are my thoughts.

A controller should exist inside a module, not outside. I know that some tutorials do that, but there is no reason to leave a controller outside a module (I saw that you fix that in a later article).

The problem so far is the:

```
$('#input').val("");
```

I can understand that you want to use jQuery (but maybe there is no need for a todo app) but you shouldn't manipulate your DOM inside a controller, not even for the sake of a tutorial. That is a really really bad practice that everyone should avoid. There is an easy way to clean your input:

```
$scope.formData.text = "";
```

Since the input is two-way binded to that primitive, you can assign an empty string to remove the text of the input.

Other than that, good tutorial, I'll keep reading it :)

3 ^ | v • Reply • Share ›



Chris Sevilleja Bartender ➔ Jesus Rodriguez • 6 months ago

Hey Jesus, thank you very much for your input. You are completely right and I realized my mistake just this past week when I was working on the other to-do tutorials. The current code for the demo utilizes `$scope.formData.text = ""`; but I haven't updated the tutorial. I'll definitely point it out to make sure that people understand that's the best way to do it. Thanks again.

^ | v • Reply • Share ›



Jo Rafali ➔ Chris Sevilleja • 4 months ago

Hi Chris,

The GH repo seems to be an older version of the code as this line is still in there.

Actually I must thank Jesus: I spent a bit of time trying to figure out what this was all about as I'm new to Web Dev and not familiar with JQuery, until I read his comment and realised the code isn't the same on the repo.

Btw, thanks for the great tutorials, extremely helpful and to the point!

^ | v • Reply • Share ›



Chris Sevilleja Bartender ➔ Jo Rafali • 4 months ago

Sorry about that. Good call. The repo has been updated with the correct code. Thanks for pointing that out and thanks for reading!

^ | v • Reply • Share ›



occulti • 8 months ago

Amazing tutorial ! Thumbs up !

3 ^ | v • Reply • Share ›



Jonas H. • 2 months ago

Great tutorial! Very readable and with working code. And dogfooding is a great term :)

1 ^ | v • Reply • Share ›



Chris Sevilleja Bartender ➔ Jonas H. • 2 months ago

Agreed. I'm spreading it as much as I can.

1 ^ | v • Reply • Share ›



Yasar Hussain • 2 months ago

how would you go about allowing multiple users to create a todo list? right now it creates one that everyone can edit but i think it'd be more useful to have it so each person has their own todo list

1 ^ | v • Reply • Share ›



Vinodh • 4 months ago

Been looking all over for a angular - express - node tutorial , and this was one of the best. Thanks a lot!

1 ^ | v • Reply • Share ›



tobbbe • 5 months ago

How/why is the collection in the database named "todos"? When I define the model I name it "Todo" (`mongoose.model('Todo', ...)`). Is it the route that does it (`/api/todos`)? I tried to change the name of the Create-route but new todos are still saved in the todos-collection

Also, what is this for: `app.use(express.methodOverride());` ?

Thanks for a grrrrrrrrreat tutorial!

1 ^ | v • Reply • Share ›



Chris Sevilleja Bartender ➔ tobbbe • 5 months ago



The collection in the database took the name of the model. The route can be named anything you want, but we just made it todos for a coherent example.

methodOverride is so that express can spoof the HTTP verb. It gives it the ability to create PUT/PATCH and DELETE HTTP calls.

2 ^ | v • Reply • Share ›



tobbbe → Chris Sevilleja • 5 months ago

But we named the model "Todo"? Is it always renamed to all lowercase and adds an s at the end?

^ | v • Reply • Share ›



Chris Sevilleja Bartender → tobbbe • 5 months ago

Yes it lower cases it and makes it plural. Here is a way to force the collection name in the database.

<http://stackoverflow.com/quest...>

2 ^ | v • Reply • Share ›



tobbbe → Chris Sevilleja • 5 months ago

Ah ok! Thanks again for some awesome tutorials!

^ | v • Reply • Share ›



goldesigner • 6 months ago

Hey Chris, I am a hobbyist and I think your tutorials are fantastic! You have a knack for simplifying complex things so they are easy to grasp.

Two questions for you:

1. When I launch localhost:3000 to load the index.html page, I don't see a GET request in my Terminal for "/" or anything. When the controllers load I see a GET request for /api/todos. So why is the initial GET from the browser to node not logged?
2. Your website doesn't have any direct means to contact you. How does one contact you for a Web Dev project other than through LinkedIn, Google+ or other social sites? Or is that your preferred mode of client introduction?

1 ^ | v • Reply • Share ›



Chris Sevilleja Bartender → goldesigner • 6 months ago

1. That is an interesting question. On my machine, I don't see the initial "/" request either but I get the /api/todos. When trying that in our [Node Authentication](#) application, you are able to see the "/" request. To be honest, I'm not really sure. I'll look into it further and try to find out why.

2. Good point. My email is kind of hidden through a couple links on my Google+. I'll add an email link to the about page after this so thanks for pointing that out.

You can contact me directly via email at chris@scotch.io.

^ | v • Reply • Share ›



Nestor Rodriguez • 8 months ago

Good stuff. Learned quite a lot through this tutorial and already look forward to that next article on using Passport for auth.

1 ^ | v • Reply • Share ›



Chris Sevilleja Bartender → Nestor Rodriguez • 8 months ago

Good to hear. I'm going to be making an entire series on this. Next is auth. Then [socket.io](#). Then whatever else comes up.

^ | v • Reply • Share ›



TeddyJenkins → Chris Sevilleja • 8 months ago

that would be super awesome. this is one of the best ANGULAR tuts out and believe me i've SEARCHED

4 ^ | v • Reply • Share ›



Chris Sevilleja Bartender → TeddyJenkins • 8 months ago

Thanks for the validation! I was definitely in the same boat. Couldn't find any really solid tuts for implementation and the Angular docs could be much

implementation and the angular todo code seems so much better. Might as well make tuts as I learn!

1 ^ | v • Reply • Share ›



Nestor Rodriguez → Chris Sevilleja • 8 months ago

Chris, sounds solid! I think you definitely should go ahead and write some more tutorials as you learn -- sure it'll help a lot of devs out there trying to get into Angular, myself included. BTW, have you tried out this app with Angular 1.2? Deleting todos doesn't work. I think this has to do with this change: <http://goo.gl/D6lw0Z> but I haven't figured out how to fix it yet. Any ideas?

^ | v • Reply • Share ›



Chris Sevilleja Bartender → Nestor Rodriguez • 8 months ago

Glad to hear they help. Just released the next one (<http://scotch.io/tutorials/jav...> for app organization and file structure. It'll help move our app in the right direction.

As for 1.2, I haven't quite figured out what the issue is yet. There's a lot of gobbledygook to run through on that page you linked. Thanks for the link btw, that will definitely help.

I'll keep you posted.

^ | v • Reply • Share ›



Nestor Rodriguez → Chris Sevilleja • 8 months ago

Actually, just figured out what the issue was. It turns out that one of the breaking changes in Angular 1.2.0-rc2 (currently latest stable build) is the introduction of private properties on the scope chain. This potentially breaks a lot of apps that store data in document-oriented databases such as MongoDB. For now, I'm using version 1.2-rc3, which is the previous build version and hadn't introduced this change yet. But the Angular changelog mentions that sensitive APIs should be wrapped in a closure/controller instead (link: <http://goo.gl/cHzYLO>). Lots of people out there with broken apps because of this change.

Excited about the tut on file organisation, btw. Will check that one out soon.

^ | v • Reply • Share ›



Chris Sevilleja Bartender → Nestor Rodriguez • 8 months ago

Good find! That'll definitely help fix up the to do app. Thanks for the info. I was debating on doing an "Upgrade your Angular App from 1.0.8 to 1.2" article so this info can be spread to the world. (I'll be sure to credit you for the work!)

^ | v • Reply • Share ›



Nestor Rodriguez → Chris Sevilleja • 8 months ago

Sounds great Chris! I'm interested in this new suggested approach I mentioned in my previous post, so if I put some code together to cover that before you release your Upgrade to 1.2 article, I'll let you know.

^ | v • Reply • Share ›



Chris Sevilleja Bartender → Nestor Rodriguez • 8 months ago


Awesome thanks! That'll probably be late next week. I have a couple Angular/Node articles to do before I get back to the To Do app.


^ | v • Reply • Share ›





Rajanikanth • 2 hours ago

Thank you for Your tutorial.


- 

^ | v • Reply • Share ›
- 

Javier Rosas • a day ago
Hi Scotch. Can you make a tutorial of how to integrate Angular.js + Twitter Bootstrap?
^ | v • Reply • Share ›
- 


Feldspar Glockenspiel • 3 days ago
Six months later and the code is already stale. It's nearly impossible to find useful tutorials as the node / express / yeoman / flavor of the week will break APIs on a whim. Not that this wasn't a nice tutorial, but its worthless now to newcomers. :-(
^ | v • Reply • Share ›
- 


Chris Sevilleja Bartender → Feldspar Glockenspiel • 2 days ago
What do you feel needs to be updated? The only thing I can see is that it should be updated for Express 4.0 which only requires about 2 lines of changes. (removing app.configure() and adding the morgan package).


At a quick glance, upgrading Angular to the 1.2 version also doesn't require changes. If you have any updates, I'd be happy to update the article.
^ | v • Reply • Share ›
- 

David Coffee Sivocha • 14 days ago
Just a usability thought. Instead of having ng-click on the button, you would be better off adding that as ng-submit on the form tag.

This would allow your users to submit the form by enter as well, which means you could do away with the button etc.


Other than that, good tutorial!
^ | v • Reply • Share ›
- 


Bryan Christophe Green • 15 days ago
This code doesn't work as expected (the data brought back from the DB is written to the console, not the \$scope, for one thing). The code in the Github repo is totally different from the code in this tut-- which I didn't realize until I spent a couple of hours debugging everything. All in all a good tutorial, though, as long as this code gets updated.
^ | v • Reply • Share ›
- 

HHH • 17 days ago
Hi Chris.. I feel this is an awesome example of writing the code in a neat way. But I think it is a bit difficult to understand it for a starter. If you can elaborate more on the application structure and the connections and how we should split the code across different directory structure, it'll be really helpful or else you can suggest me any similar readings with more basics explained..
^ | v • Reply • Share ›
- 


Chris Sevilleja Bartender → HHH • 17 days ago
Hello thanks for your comment. You could try our other MEAN starter tutorial: <http://scotch.io/bar-talk/setting-up-a-mean-stack>. That gives some more overview on starting applications like this.

I would also suggest that if you are having trouble with the structure of the code, take a look at the GitHub repo at: <https://github.com/scotch-io/n...>

You can see the exact folder structure and code from each file. You can even download the entire code there and test it.
^ | v • Reply • Share ›
- 

Justin • a month ago
I'm new to web development and Java Script. I'm trying to figure out where the values (`{todos.length}`, `{todo.text}`) come from in the index.html. Can someone explain this to me please? Thanks.
^ | v • Reply • Share ›
- 

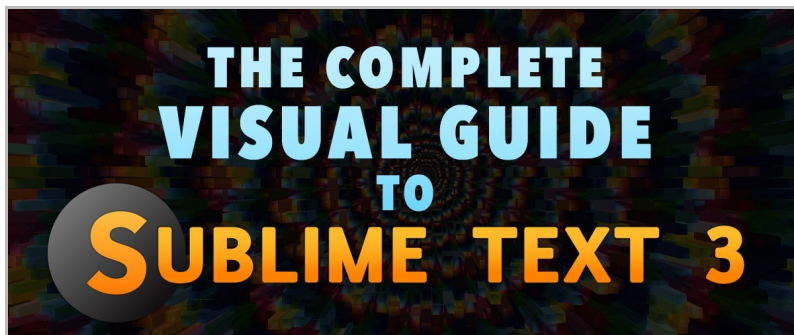
Guest • a month ago

 Creators of JIRA,
Confluence, and Stash

AdChoices

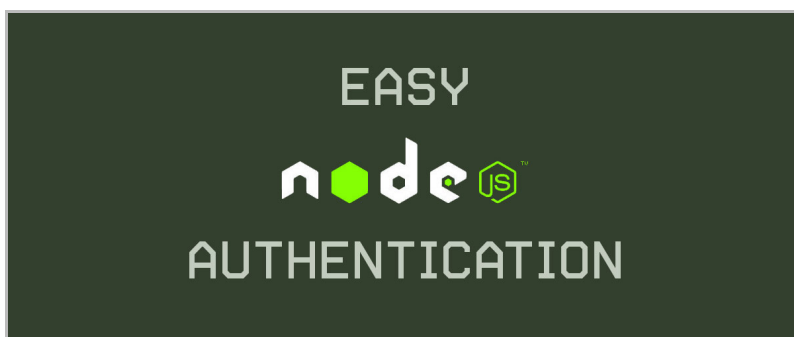
Start Your Free Trial

Popular Series



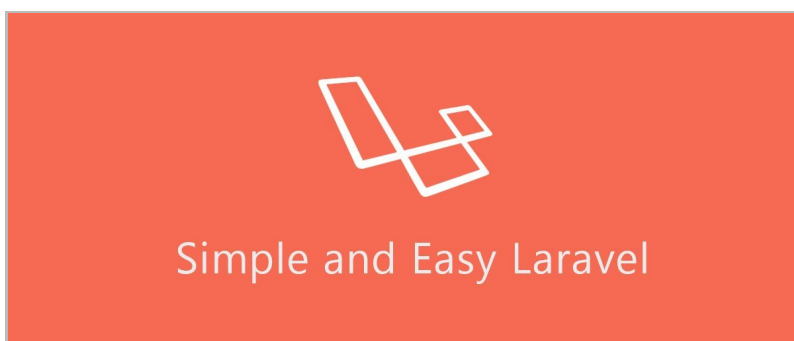
(/series/the-complete-visual-guide-to-sublime-text-3)

The Complete Visual Guide to Sublime Text 3 (/series/the-complete-visual-guide-to-sublime-text-3)



(/series/easy-node-authentication)

Easy Node Authentication (/series/easy-node-authentication)



(/series/simple-laravel)

Simple and Easy Laravel (/series/simple-laravel)