



## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

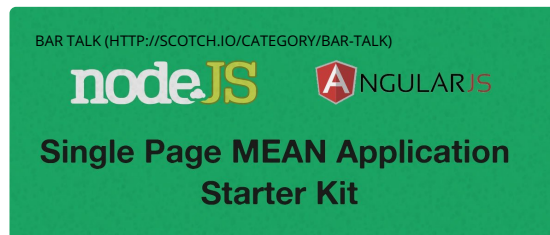
Projects  
(/work)

Shop  
(http://shop.scotch.io)

About  
(/about)

Search Bar ▾ Drinks are on us...

Like 2.9K Follow 11.3K followers



# Setting Up a MEAN Stack Single Page Application



Chris Sevilleja  
(http://scotch.io/author/chris) January 24, 2014 105 Comments angularJS  
(http://scotch.io/tag/angular-js), MEAN  
(http://scotch.io/tag/mean), node.js  
(http://scotch.io/tag/node-js), single page  
app (http://scotch.io/tag/single-page-app)

158  
✈

29  
g+

115  
f

<> VIEW CODE (HTTPS://GITHUB.COM/SCOTCH-IO/STARTER-NODE-ANGULAR)



(http://stats.buysellads.com/click.go?z=1291897&b=5158906&g=&s=&sw=1920&sh=1200&br=firefox,33,mac&r=0.2270820023984056&link=http://www.httpdebugger.com/)



Buy & Sell Code  
PHP

(http://stats.buysellads.com/click.go?z=1291908&b=5062428&g=&s=&sw=1920&sh=1200&br=firefox,33,mac&r=0.23172369859826725&link=https://codeclerks.com/?utm\_source=bsa&utm\_medium=cpm&utm\_term=&utm\_content=&utm\_campaign=scotch.io)



(http://stats.buysellads.com/click.go?z=1291908&b=5192850&g=&s=&sw=1920&sh=1200&br=firefox,33,mac&r=0.8008544336790738&link=http://www.host1plus.com/vps-hosting/?utm\_source=buysell&utm\_medium=banner&utm\_campaign=125)



(http://stats.buysellads.com/click.go?z=1291908&b=5165288&g=&s=&sw=1920&sh=1200&br=firefox,33,mac&r=0.4399793033043733&link=http://jobety.com/?utm\_source=scotch-io&utm\_medium=fixed&utm\_content=125x125)



(http://stats.buysellads.com/click.go?z=1291908&b=5162192&g=&s=&sw=1920&sh=1200&br=firefox,33,mac&r=0.7963770389177398&link=http://www.socialstrap.net/?utm\_source=scotch-io&utm\_medium=fixed&utm\_content=125x125)



(http://stats.buysellads.com/click.go?z=1291908&b=5157086&g=&s=&sw=1920&sh=1200&br=firefox,33,mac&r=0.15934306928260622&link=http://www.webrtcworld.com/default.aspx)



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)


About  
(/about)

Beginning an application from scratch can sometimes be the hardest thing to do. Staring at an empty folder and a file with no code in it yet can be a very daunting thing.

In today's tutorial, we will be looking at the **starting setup** for a Node.js, AngularJS, MongoDB, and Express application (otherwise known as MEAN). I put those in the wrong order, I know.

This will be a starting point for those that want to learn how to begin a MEAN stack application. Projects like mean.io (<http://mean.io>) and meanjs.org (<http://meanjs.org>) are more fully fledged MEAN applications with many great features you'd want for a production project.

You will be able to start from absolute scratch and create a basic application structure that will allow you to build any sort of application you want.

 This article has been updated to work with Express 4.0

## What We'll Be Building

A lot of the applications we've dealt with so far had a specific function, like our Node and Angular To-Do Single Page Application (<http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular>). We are

## POPULAR ON SCOTCH

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

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

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

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

ExpressJS 4.0: New Features and Upgrading from 3.0  
(<http://scotch.io/bar-talk/expressjs-4-0-new-features-and-upgrading-from-3-0>)

Simple and Easy Laravel Login Authentication  
(<http://scotch.io/tutorials/simple-and-easy-laravel-login-authentication>)



**Gartner Magic Quadrant:**  
Große Veränderungen in Sachen BI  
[Zum Download](#)



## GET OUR LATEST ARTICLES

Your Secret Electronic Address

SUBSCRIBE

Like  2.9k Follow  11.3K followers



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

going to step away from that and just a good old getting started application.

This will be very barebones but hopefully it will help you set up your applications. Let's just call it a **starter kit**.

## Application Requirements

- Single Page Application
- Node.js backend with Express and MongoDB
- AngularJS frontend
- Modular Angular components (controllers, services)
- Good application structure so our app can grow

This tutorial will be more based on application structure and creating a solid foundation for single page MEAN stack applications. For more information on CRUD, authentication, or other topics in MEAN apps we'll make sure to write other tutorials to fill those gaps.

## The Backend Node, MongoDB, Express

Three letters out of the MEAN stack will be handled on the backend, our server. We will **create our server**, **configure our application**, and **handle application routing**.

## Tools Required

We will need Node (<http://nodejs.org/>) and to make our lives easier, we'll use bower (<http://bower.io>) to pull in all our dependencies.

**Bower** isn't really necessary.



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

You could pull in all the files we need yourself (bootstrap, angular, angular-route), but bower just gets them all for you! For more info, read our **guide on bower (<http://scotch.io/bar-talk/manage-front-end-resources-with-bower>)** to get a better understanding.

## Application Structure

By the end of this tutorial, we will have a basic application structure that will help us develop our Node backend along with our Angular frontend. Here's what it will look like.

```
- app
  ----- models/
  ----- nerd.js <!-- the nerd model
  ----- routes.js
- config
  ----- db.js
- node_modules <!-- created by npm install
- public <!-- all frontend and angular stuff
  ----- css
  ----- js
  ----- controllers <!-- angular control
  ----- services <!-- angular services
  ----- app.js <!-- angular application
  ----- appRoutes.js <!-- angular routes
  ----- img
  ----- libs <!-- created by bower install
  ----- views
  ----- home.html
  ----- nerd.html
  ----- geek.html
  ----- index.html
- .bowerrc <!-- tells bower where to put files
- bower.json <!-- tells bower which files to install
- package.json <!-- tells npm which package to install
- server.js <!-- set up our node application
```

AD



We'll be filling in our files into folder



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)  
Shop  
(<http://shop.scotch.io>)

About  
(/about)

structure. All backend work is done in `server.js`, `app`, and `config` while all the frontend is handled in the `public` folder.

## Starting Our Node Application

### package.json

All Node applications will start with a `package.json` file so let's begin with that.

```
{
  "name": "starter-node-angular",
  "main": "server.js",
  "dependencies": {
    "express" : "~4.5.1",
    "mongoose" : "~3.8.0",
    "body-parser" : "~1.4.2",
    "method-override" : "~2.0.2"
  }
}
```

That's it! Now our application will know that we want to use Express and Mongoose.

**Note on Express 4** Since Express 4.0, `body-parser` and `method-override` are their own modules, which is why we have to include them here. For more information, here's our guide to Express 4 (</bar-talk/expressjs-4-0-new-features-and-upgrading-from-3-0>). **Express** is a Node.js web application framework that will help us create our application. **Mongoose** is a MongoDB ORM that will help us communicate with our MongoDB database.

## Install Node Modules

To install the dependencies we just setup, just go into your console and type:

```
npm install
```

 You'll see your



scotch.io  
code on the rocks  
(<http://scotch.io>)

#### LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

#### OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

application working to bring in those modules into the `node_modules` directory that it creates.

Now that we have those, let's set up our application in `server.js`.

## Setting Up Our Node Application

`server.js`

Since this is our starter kit for a single page MEAN application, we are going to keep this simple. The entire code for the file is here and it is commented for help understanding.



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)  
Shop  
(<http://shop.scotch.io>)  
About  
(</about>)

```
// server.js

// modules =====
var express      = require('express');
var app          = express();
var bodyParser   = require('body-parser');
var methodOverride = require('method-override')

// configuration =====

// config files
var db = require('./config/db');

// set our port
var port = process.env.PORT || 8080;

// connect to our mongoDB database
// (uncomment after you enter in your own crede
// mongoose.connect(db.url);

// get all data/stuff of the body (POST) param
// parse application/json
app.use(bodyParser.json());

// parse application/vnd.api+json as json
app.use(bodyParser.json({ type: 'application/vr

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true

// override with the X-HTTP-Method-Override hea
app.use(methodOverride('X-HTTP-Method-Override'

// set the static files location /public/img wi
app.use(express.static(__dirname + '/public'));

// routes =====
require('./app/routes')(app); // configure our

// start app =====
// startup our app at http://localhost:8080
app.listen(port);

// shoutout to the user
console.log('Magic happens on port ' + port);

// expose app
exports = module.exports = app;
```

We have now pulled in our modules, configured our application for things like database, some express settings, routes, and then started our server. Notice how we didn't pull in mongoose here. There's no need for it yet. We will be using it in our model that we will define soon.

Let's look at config, a quick model, and routes since we haven't created those yet. Those will be the last things that the backend side of our application needs.



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)

Shop  
(<http://shop.scotch.io>)

About  
(</about>)

## Config config/

I know it doesn't seem like much now since we only are putting the `db.js` config file here, but this was more for demonstration purposes. In the future, you may want to add more config files and call them in `server.js` so this is how we will do it.

```
// config/db.js
module.exports = {
  url : 'mongodb://localhost/stencil-dev'
}
```

Now that this file is defined and we've called it in our `server.js` using `var db = require('./config/db');`, you can call any items inside of it using `db.url`.

For getting this to work, you'll want a local MongoDB database installed or you can just grab a quick one off services like Modulus (<http://modulus.io>) or Mongolab (<http://mongolab.com>). Just go ahead and create an account at one of those, create a database with your own credentials, and you'll be able to get the URL string to use in your own config file.

Next up, we'll create a quick Mongoose model so that we can define our Nerds in our database.

## Nerd Model app/models/nerd.js

This will be all that is required to create records in our database. Once we define our Mongoose model, it will let us handle creating, reading, updating, and deleting our nerds.

Let's go into the `app/models/nerd.js` file and add the following:





scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

```
// app/models/nerd.js
// grab the mongoose module
var mongoose = require('mongoose');

// define our nerd model
// module.exports allows us to pass this to oth
module.exports = mongoose.model('Nerd', {
  name : {type : String, default: ''}
});
```

This is where we will use the Mongoose module and be able to define our Nerd model with a name attribute with data type `String`. If you want more fields, feel free to add them here. Read up on the Mongoose docs (<http://mongoosejs.com/docs/guide.html>) to see all the things you can define.

Let's move onto the routes and use the model we just created.

## Node Routes `app/routes.js`

In the future, you can use the **app** folder to add more models, controllers, routes, and anything backend (Node) specific to your app.

Let's get to our routes. When creating a single page application, you will usually want to separate the functions of the backend application and the frontend application as much as possible.

## Separation of Routes

To separate the duties of the separate parts of our application, we will be able to define as many routes as we want for our Node backend. This could include API routes or any other routes of that nature.

We won't be diving into those since we're not really handling creating an API or doing CRUD in this tutorial, but just know that this is where you'd handle those routes.



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
([/tutorials](#))

Series  
([/series](#))

Bar Talk  
([/bar-talk](#))

Quick Tips  
([/quick-tips](#))

## OUR STUFF

Projects  
([/work](#))

Shop  
(<http://shop.scotch.io>)

About  
([/about](#))

We've commented out the place to  
put those routes here.

```
// app/routes.js

// grab the nerd model we just created
var Nerd = require('./models/nerd');

module.exports = function(app) {

  // server routes =====
  // handle things like api calls
  // authentication routes

  // sample api route
  app.get('/api/nerds', function(req, res) {
    // use mongoose to get all nerds in
    Nerd.find(function(err, nerds) {

      // if there is an error retrieve
      // nothing after
      if (err)
        res.send(err);

      res.json(nerds); // return all
    });
  });

  // route to handle creating goes here (
  // route to handle delete goes here (ap

  // frontend routes =====
  // route to handle all angular requests
  app.get('*', function(req, res) {
    res.sendFile('./public/views/index.
  });
};
```

This is where you can handle your  
API routes. For all other routes (\*),  
we will send the user to our  
frontend application where Angular  
can handle routing them from there.

## Backend Done!

Now that we have everything we  
need for our server to get setup! At  
this point we can **start our server**,  
\*\*send a user the Angular app  
( `index.html` ), and handle 1 API  
route to get all the nerds.

Let's create that `index.html` file so  
that we can test out our server.

## Create an Index View

**File** `public/views/index.html`



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)  
Shop  
(<http://shop.scotch.io>)

About  
(</about>)

Let's just open up this file and add some quick text so we can test our server.

```
<!-- public/views/index.html -->
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">

  <title>Starter MEAN Single Page Application

</head>
<body>

  we did it!

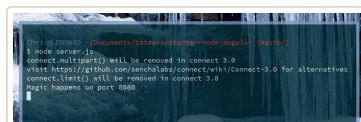
</body>
</html>
```

## Test Our Server

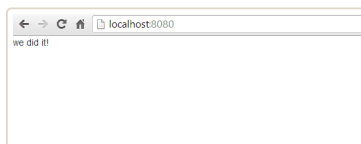
With all the backend (and a tiny frontend piece) in place, let's start up our server. Go into your console and type:

```
$ node server.js
```

Now in our console we have:



(<http://scotch.io/wp-content/uploads/2014/01/node-server-start.png>) Now we can go into our browser and see `http://localhost:8080` in action.



(<http://scotch.io/wp-content/uploads/2014/01/node-server-hello.png>) So simple, and yet so beautiful. Now let's get to the frontend single page AngularJS stuff.



scotch.io  
code on the rocks  
(<http://scotch.io>)

#### LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

#### OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

## The Frontend

### AngularJS

With all of our backend work in place, we can focus on the frontend. Our Node backend will send any user that visits our application to our `index.html` file since we've defined that in our **catch-all route** (`app.get('*')`).

The frontend work will require a few things:

- Files and libraries brought in by Bower
- Angular application structure (controllers, services)
- We will create 3 different pages (Home, Nerds, Geeks)
- Handle Angular routes using `ngRoute` (<http://docs.angularjs.org/api/ngRoute>) so there are no page refreshes
- Make it pretty with Bootstrap

## Bower and Pulling in Components

We will need certain files for our application like bootstrap and of course angular. We will tell bower to grab those components for us.

Bower is a great frontend tool to manager your frontend resources. You just specify the packages you need and it will go grab them for you. Here's an article on getting started with bower (<http://scotch.io/bar-talk/manage-front-end-resources-with-bower>).

First we will need Bower **installed** on our machine. Just type in `npm install -g bower` into your console.

After you have done that, you will now have access to bower globally



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

on your system. We will need 2 files to get Bower working for us ( `.bowerrc` and `bower.json` ). We'll place both of these in the root of our document.

**.bowerrc** will tell Bower where to place our files:

```
{  
  "directory": "public/libs"  
}
```

**bower.json** is similar to `package.json` and will tell Bower which packages are needed.

```
{  
  "name": "starter-node-angular",  
  "version": "1.0.0",  
  "dependencies": {  
    "bootstrap": "latest",  
    "font-awesome": "latest",  
    "animate.css": "latest",  
    "angular": "latest",  
    "angular-route": "latest"  
  }  
}
```

Let's run it! In your console, in the root of your application, type:

`bower install` You can see bower pull in all the files we needed and now we have them in `public/libs` !

Now we can get down to business and work on our Angular stuff.

## Setting Up Our Angular Application

For our Angular application, we will want:



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
([tutorials](#))

Series  
([series](#))

Bar Talk  
([bar-talk](#))

Quick Tips  
([quick-tips](#))

## OUR STUFF

Projects  
([work](#))  
Shop  
(<http://shop.scotch.io>)

About  
([about](#))

- 2 different pages (Home, Nerds)
- A different Angular **controller** for each
- An Angular **service** for Nerds
- No page refresh when switching pages

Let's create the files needed for our Angular application. This will be done in `public/js`. Here is the application structure for our frontend:

```
- public
  ---- js
    ----- controllers
    ----- MainCtrl.js
    ----- NerdCtrl.js
    ----- services
    ----- NerdService.js
    ----- app.js
    ----- appRoutes.js
```

Once we have created our **controllers**, **services**, and **routes**, we will combine them all and inject these modules into our main `app.js` file to get everything working together.

## Angular Controllers

We won't go too far in depth here so let's just show off all three of our controllers and their code.

```
// public/js/controllers/MainCtrl.js
angular.module('MainCtrl', []).controller('MainCtrl', function($scope) {

    $scope.tagline = 'To the moon and back!';

});
```



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)

Shop  
(<http://shop.scotch.io>)

About  
(</about>)

```
// public/js/controllers/NerdCtrl.js
angular.module('NerdCtrl', []).controller('NerdCtrl', function($scope) {
    $scope.tagline = 'Nothing beats a pocket pr
});
```

Of course in the future you will be doing a lot more with your controllers, but since this is more about application setup, we'll move onto the services.

## Angular Services

This is where you would use `$http` or `$resource` to do your API calls to the Node backend from your Angular frontend.

```
// public/js/services/NerdService.js
angular.module('NerdService', []).factory('NerdService', function() {
    return {
        // call to get all nerds
        get : function() {
            return $http.get('/api/nerds');
        },

        // these will work when more API
        // call to POST and create a new nerd
        create : function(nerdData) {
            return $http.post('/api/nerds', nerdData);
        },

        // call to DELETE a nerd
        delete : function(id) {
            return $http.delete('/api/nerds/' + id);
        }
    };
});
```

That's it for our services. The only function that will work in that `NerdService` is the `get` function. The other two are just placeholders and they won't work unless you define those specific routes in your `app/routes.js` file. For more on building APIs, here's a tutorial for [Building a RESTful Node API](#)



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

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

These services will call our Node backend, retrieve data in JSON format, and then we can use it in our Angular controllers.

## Angular Routes

Now we will define our Angular routes inside of our `public/js/appRoutes.js` file.

```
// public/js/appRoutes.js
angular.module('appRoutes', []).config(['$r
$routeProvider

    // home page
    .when('/', {
        templateUrl: 'views/home.html',
        controller: 'MainController'
    })

    // nerds page that will use the NerdCon
    .when('/nerds', {
        templateUrl: 'views/nerd.html',
        controller: 'NerdController'
    });

    $locationProvider.html5Mode(true);

}]);
```

Our Angular frontend will use the template file and inject it into the `<div ng-view></div>` in our `index.html` file. It will do this without a page refresh which is exactly what we want for a single page application.

For more information on Angular routing and templating, check out our other tutorial: Single Page Apps with AngularJS (<http://scotch.io/tutorials/javascript/single-page-apps-with-angularjs-routing-and-templating>).

## Updated View Files





scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)  
Shop  
(<http://shop.scotch.io>)

About  
(/about)

With all of the Angular routing ready to go, we just need to create the view files and then the smaller template files ( `home.html` , `nerd.html` , and `geek.html` ) will be injected into our `index.html` file inside of the `<div ng-view></div>` .

Notice in our `index.html` file we are calling the resources we pulled in using bower.

```
<!-- public/index.html -->
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <base href="/">

  <title>Starter Node and Angular</title>

  <!-- CSS -->
  <link rel="stylesheet" href="libs/bootstrap">
  <link rel="stylesheet" href="css/style.css"

  <!-- JS -->
  <script src="libs/angular/angular.min.js"><
  <script src="libs/angular-route/angular-rou

  <!-- ANGULAR CUSTOM -->
  <script src="js/controllers/MainCtrl.js"></
  <script src="js/controllers/NerdCtrl.js"></
  <script src="js/services/NerdService.js"></
  <script src="js/appRoutes.js"></script>
  <script src="js/app.js"></script>
</head>
<body ng-app="sampleApp" ng-controller="NerdCor
<div class="container">

  <!-- HEADER -->
  <nav class="navbar navbar-inverse">
    <div class="navbar-header">
      <a class="navbar-brand" href="/">St
    </div>

    <!-- LINK TO OUR PAGES. ANGULAR HANDLES
    <ul class="nav navbar-nav">
      <li><a href="/nerds">Nerds</a></li>
    </ul>
  </nav>

  <!-- ANGULAR DYNAMIC CONTENT -->
  <div ng-view></div>

</div>
</body>
</html>
```



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials

(/tutorials)

Series

(/series)

Bar Talk

(/bar-talk)

Quick Tips

(/quick-tips)

## OUR STUFF

Projects

(/work)

Shop

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

About

(/about)

```
<!-- public/views/home.html -->

<div class="jumbotron text-center">
  <h1>Home Page 4 Life</h1>

  <p>{{ tagline }}</p>
</div>
```

```
<!-- public/views/nerd.html -->

<div class="jumbotron text-center">
  <h1>Nerds and Proud</h1>

  <p>{{ tagline }}</p>
</div>
```

## Making It All Work Together

We have defined our resources, controllers, services, and routes and included the files in our `index.html`. Now let's make them all work together.

Let's set up our Angular app to use all of our components. We will use dependency injection and set up our Angular application.

```
// public/js/app.js
angular.module('sampleApp', ['ngRoute', 'appRou
```

## Conclusion

Now we have an application that has a Node.js backend and an AngularJS frontend. We can use this foundation to build any sort of application moving forward. We can add authentication (<http://scotch.io/series/easy-node-authentication>) and CRUD functionality to create a good application.



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)

Shop  
(<http://shop.scotch.io>)

About  
(</about>)

## Next Steps

Moving forward, I'd encourage you to take this and see if it fits your needs. The point of this was to have a foundation for starting applications so that we aren't reinventing the wheel every time we start a new project.

This is a very barebones example and for something more in depth, I'd encourage people to take a look at [mean.io](http://mean.io) (<http://mean.io>) for a more in depth starter application.

Check out the Github repo (<https://github.com/scotch-io/starter-node-angular>) for this project and take from it what you need. Sound off in the comments if you have any questions about how to expand this into your own applications.

## Starter Kit

We've put this tutorial together as a starter kit at the Github repo (<https://github.com/scotch-io/starter-node-angular>). We'll keep adding features to it on request and any updates we think will be helpful for applications.

Hopefully it will be a good foundation for any sort of Single Page MEAN Stack Application.

## To Use the Starter App

1. Download the code  
(<https://github.com/scotch-io/starter-node-angular/archive/master.zip>)
2. Install the npm modules: `npm install`
3. Install the bower components:  
`bower install`
4. Start the server: `node server.js`
5. Visit the application in your browser at



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

`http://localhost:8080`

6. Use this starter kit to build any application you need.

Further Reading: When building MEAN stack apps, the backend Node application will usually be an API that we build. This will allow the Angular frontend to consume our API that we built through Angular services. The next step is to hash out building a Node API. This next tutorial will teach us that and then we can go further in depth of how to build the frontend Angular application to consume our new API.

BUILD A RESTFUL API USING NODE AND EXPRESS 4 • (/TUTORIALS/JAVASCRIPT/BUILD-A-RESTFUL-API-USING-NODE-AND-EXPRESS-4)

## Want More MEAN?

This article is part of our Getting MEAN (/series/getting-mean) series. Here are the other articles.

- Setting Up a MEAN Stack Single Page Application
- Build a RESTful API Using Node and Express 4 (/tutorials/javascript/build-a-restful-api-using-node-and-express-4)
- Using GruntJS in a MEAN Stack Application (/tutorials/javascript/using-gruntjs-in-a-mean-stack-application)

**Edit #1** (7/8/14): Updated article for Express 4 support. Thanks to Caio Mariano (<https://github.com/caionitro>) for the help.

**Edit #1** (10/12/14): Updated article to add Nerd model and make everything clearer.

## Further Reading

Thirsty for more? Here are some related articles to keep you learning.



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



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



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



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)

Shop  
(<http://shop.scotch.io>)

About  
(</about>)

<a href="/expressjs-4-0-new-features-and-upgrading-from-3-0">/expressjs-4-0-new-features-and-upgrading-from-3-0</a>	<a href="/javascript/creating-a-single-page-todo-app-with-node-and-angular">/javascript/creating-a-single-page-todo-app-with-node-and-angular</a>	<a href="/javascript/using-gruntjs-in-a-mean-stack-application">/javascript/using-gruntjs-in-a-mean-stack-application</a>
ExpressJS 4.0: New Features and Upgrading from 3.0 ( <a href="http://scotch.io/bar-talk/expressjs-4-0-new-features-and-upgrading-from-3-0">http://scotch.io/bar-talk/expressjs-4-0-new-features-and-upgrading-from-3-0</a> )	Creating a Single Page Todo App with Node and Angular ( <a href="http://scotch.io/tutorials/javascript/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</a> )	GruntJS in a MEAN Stack Application ( <a href="http://scotch.io/tutorials/javascript/using-gruntjs-in-a-mean-stack-application">http://scotch.io/tutorials/javascript/using-gruntjs-in-a-mean-stack-application</a> )

</> VIEW CODE ([HTTPS://GITHUB.COM/SCOTCH-IO/STARTER-NODE-ANGULAR](https://github.com/scotch-io/starter-node-angular))

158



29



115



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

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



scotch.io  
code on the rocks  
(http://scotch.io)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)  
Shop  
(http://shop.scotch.io)

About  
(/about)

/chris)

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

Follow @sevilayha

♥ VIEW MY ARTICLES (HTTP://SCOTCH.IO/AUTHOR/CHRIS)

## Stay Connected With Us

hover these for magic (/tutorials  
/css/css3-hidden-social-buttons)

(HTTP://(HTTP://(HTTP://(HTTP://  
/FEED) /SCOTCH/SCOTCH/CI/B/1138!  
SUBSCR FOLLOW LIKE +1 /+SCOTC



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

Your Secret Electronic Address

SUBSCRIBE



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(</tutorials>)

Series  
(</series>)

Bar Talk  
(</bar-talk>)

Quick Tips  
(</quick-tips>)

## OUR STUFF

Projects  
(</work>)

Shop  
(<http://shop.scotch.io>)

About  
(</about>)



scotch.io  
code on the rocks  
(<http://scotch.io>)

## LEARN

Tutorials  
(/tutorials)

Series  
(/series)

Bar Talk  
(/bar-talk)

Quick Tips  
(/quick-tips)

## OUR STUFF

Projects  
(/work)

Shop  
(<http://shop.scotch.io>)

About  
(/about)

# USE THE FORCE, LUKE

(MORE ARTICLES)

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

Nov 7

## Creating a Single Page Todo App



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

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

(<http://scotch.io/bar-talk/bootstrap-3-tips-and-tricks-you-still-might-not-know>)

Sep 30

## Bootstrap 3 Tips and Tricks You Still



**Bootstrap**  
(<http://scotch.io/bar-talk/bootstrap-3-tips-and-tricks-you-still-might-not-know>)

**3-tips-and-tricks-you-still-might-not-know**  
(<http://scotch.io/tutorials/javascript/submitting-ajax-forms-the-angularjs-way>)

Nov 18

## Submitting AJAX Forms: The



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

**the-angularjs-way**





scotch.io  
code on the rocks  
(<http://scotch.io>)



(</tag/angular.js>)  
AngularJS



(</tag/node.js>)  
Node.js



(</tag/jquery>)  
jQuery



(</tag/mongo>)  
MongoDB



(</tag/express>)  
Express.js



(</tag/laravel>)  
Laravel



(</tag/css3>)  
CSS3



(</tag/bootstrap>)  
Bootstrap



(</tag/sublime-text>)  
Sublime



(</tag/vagrant>)  
vagrant

[Advertise with Us \(/advertise\)](/advertise)

[Write for Us \(/write-for-us\)](/write-for-us)

© Scotch.io 2014

[Contact Us \(/contact-us\)](/contact-us)

Hosted by (<https://www.digitalocean.com/?refcode=7a59e9361ab7>)

## LEARN

[Tutorials](/tutorials)

(</tutorials>)

[Series](/series)

(</series>)

[Bar Talk](/bar-talk)

(</bar-talk>)

[Quick Tips](/quick-tips)

(</quick-tips>)

## OUR STUFF

[Projects](/work)

(</work>)

[Shop](http://shop.scotch.io)

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

[About](/about)

(</about>)