Link: https://hackhands.com/finishing-Angular-TODO-application-deploying-production/

Posted on June 30th 2015

# Finishing Angular TODO application and deploying to production

## TL;DR

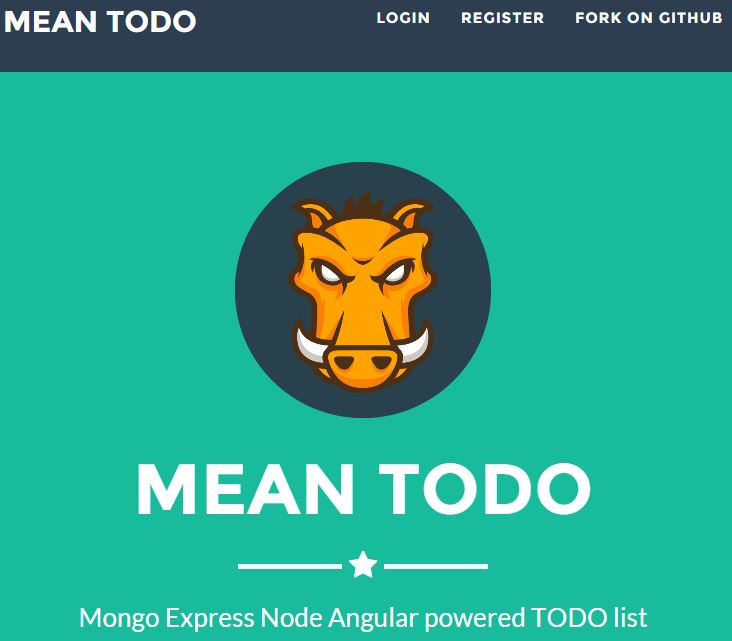
This is a fourth post (first one is about [getting started on the MEAN stack](https://hackhands.com/how-to-get-started-on-the-mean-stack/), the second one is all about[Node.js and Express](https://hackhands.com/delving-node-js-express-web-framework/) and the third one is about [MongoDB, Mongoose and Passport Authentication](https://hackhands.com/mongodb-crud-mvc-way-with-passport-authentication/)) in a series of posts which will teach you how to take the advantage of the MEAN stack in becoming a full-stack JavaScript developer.

In this post, which completes the MEAN circle, you'll learn how [AngularJS](https://angularjs.org/) works with Node.js, Express and MongoDB - and all that in a MVC way. We'll dive directly in the code, by continuing from the [previous tutorial](https://hackhands.com/mongodb-crud-mvc-way-with-passport-authentication/) and implement the TODO application, and I'll explain the basics as we go along. Finally, you'll go over the steps needed to deploy the application to the actual server and make it available publicly.

Disclaimer: code is based on the[*MEAN.JS framework*](http://meanjs.org/)and the tutorial partly follows the structure from[*MEAN Web Development by Amos Haviv*](http://amzn.to/1cvHyth), the author of the MEAN.JS framework.

## What are we going to build?

In this tutorial you'll build the exact replica of [MEANTODO.com](http://meantodo.com/):



I'll give you the exact source code for free (as in beer - and the license is "you can do what ever you want with it as long as you don't blame me for your server going up in smoke kind of license" [think[*Richard Stallman*](http://hr.wikipedia.org/wiki/Richard_Stallman):)]) and show you how to put it online. If you're like



here are the details for getting your feet wet immediately:

* [clone the finished project from Github](https://github.com/Hitman666/MEAN_MVC_3rdTutorial.git)
* run **npm install && bower install** after cloning
* make sure that you have **mongod** running on default 27017 port
* run **node server**from the root of the application (where the **server.js** file is placed)
* the details on how to get this on production server are in the [**Deploying me gently**](https://hackhands.com/finishing-Angular-TODO-application-deploying-production/#deploy)section
* note: if you get a Mongoose unstable error, check[*this StackOverflow post*](http://stackoverflow.com/questions/24023912/avoid-unstable-releases-of-mongoose-in-npm-package-json)

### Where have you been?!!?

It's been a while since **A**ngie finally decided to join the **MEN** (MongoDB, Express, Node.js, naturally) letters of our stack. I was actually on a secret mission for NSA, and since there's No Such Agency, I can't tell you any more about it. Anyways, we're here now and let's get this AngularJS thingy under our belt. In this tutorial you'll finally learn how to get all the strings together and how AngularJS works along with the rest of the good folks in the MEN department.

### Meh, you're just too late

However, late as it may seem, you may be complaining that's too late, that Angular 2.0 is almost out and you may feel that:



Angular 2.0 will not be out still for some time, and Angular 1.4 just came out [few weeks ago](http://angularjs.blogspot.com/2015/05/angular-140-jaracimrman-existence.html) (27.5.2015). In my opinion Angular 1.x will be with us for some time still, and even though 2.0 has some "[*new cool stuff*](http://angular-tips.com/blog/2015/06/why-will-angular-2-rock/)that's totally different from the 1.x branch (no scopes, controllers, only one service*shocked*)", nevertheless you can make "magic" with it still, so you better make some use of it. If nothing else, you can still go over this and get a grasp of how it actually works - no learned knowledge goes to waste, right? (all you VB6 coders who still use it in production please raise your hands - here is one hand_rock_n_roll).

## Let's rock 'n roll

You may remember from the [first tutorial](https://hackhands.com/how-to-get-started-on-the-mean-stack/) that Angular is a frontend JavaScript framework designed to build **single-page applications** using the MVC architecture. Also, that it's built and maintained by the almighty Google. Few of the cool features we mentioned are:

* two-way data binding, which synchronizes between models and views
* extended HTML with additional attributes, which bind the JavaScript objects with HTML elements
* improved code structure
* easier testing through dependency injection (yeah, don't worry I won't "bore" you with this in this tutorial - however, if you would like to learn more about this please request it in the comments and I may come up with a tutorial on that too - sooner than part #4 was done, I promise smileyGlasses)

AngularJS surely has it's quirks, and this video is one of the funniest in pointing this out:[https://www.youtube.com/watch?v=M\_Wp-2XA9ZU](https://www.youtube.com/watch?v=M_Wp-2XA9ZU" \t "_blank) (yeah, by now you should have realized I like to tackle "tough" stuff with a dose of humor) However, as the author Shai Reznik [says](https://youtu.be/M_Wp-2XA9ZU?t=927):

Every time we want to complain or criticize, let's create a pull request instead, OK?

### Where to start?

Well, TBH, I could simply point you to my notes from a free course from [CodeSchool](http://www.nikola-breznjak.com/blog/javascript/angular/angular-codeschool-notes/) or even from Pluralsight (update link when HH post will be live) and you would get a decent understanding. Surely, I suggest that you go through the videos yourself (especially the CodeSchool one since, well, it's free and interactive), and also, if you're more experienced I would recommend you take a look at how [Yeoman](http://www.nikola-breznjak.com/blog/javascript/angular/speeding-up-angular-development-with-yeoman/)works with Angular generator.

I'm a firm believer that [Pareto principle](http://en.wikipedia.org/wiki/Pareto_principle) rocks in all areas of life, and that's why in this Angular tutorial I won't go too deep into certain parts (the tutorial is [**huge**](http://www.nikola-breznjak.com/blog/wp-content/uploads/2015/06/JohnnyBravo.gif)without it anyways), instead I'll cover the things that you'll need to build the aforementioned TODO app, and will link to other articles if you would like to learn more about the particular subject.

### Can you keep a secret?



Let me give you in on a little secret. This what you have been building so far is based on (yeah, you may have read the disclaimer, but still let me make my point :)) the awesome [MEAN.js](http://meanjs.org/) stack originated by Amos Haviv.

But, what does this mean? Well, have you heard about the [Ember framework](http://emberjs.com/)? It's a competitor, so to speak, to AngularJS - and to be honest one of the best things in Ember is the [**ember-cli**](http://www.ember-cli.com/) which is in fact a standard in the Ember community and it enables you to have a consistent folder structure. Basically, if you change jobs and come to some other project which works with Ember using the ember-cli, you will immediately know where certain things are. This is something which couldn't have been said for AngularJS until the MEAN.js came to the scene (sure, there are some other [guidelines](https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/pub), but this is a framework). With it (sure, there is also [MEAN.io](http://mean.io/) and you can learn about the difference in my post about [MEAN.io vs MEAN.js](https://hackhands.com/mean-io-vs-mean-js-deploying-latter-digitalocean/)) you can be confident that once you come to some other project which uses MEAN.js that everything will be where you expect it to be.

## Let's make this infamous TODO app

In this first section I'll show you the basic setting for an AngularJS app, and will introduce some of its core components. As mentioned before, nothing is dealt with too much deep $hit but I do link the appropriate additional materials. In the second part you'll create the CRUD (Create Read Update Delete) modules of your TODO app.

### Clone the previous project

To start, clone the code from the [third project](https://github.com/Hitman666/MEAN_MVC_3rdTutorial)and **cd** into the project directory from your command line. In case you don't have [bower](http://bower.io/) installed (you should have it if you've followed the [first tutorial](https://hackhands.com/how-to-get-started-on-the-mean-stack/)) install it with:

npm install bower -g

Next, open up the existing file **.bowerrc** and change it to look like this:

{

"directory": "public/js"

}

Important note for those who followed the book too: you need to put double quotes on both directory and public/js. The following command will install AngularJS in **public/js** folder:

bower install angular

### AngularJS setup

Now that you've installed AngularJS, you need to include it in the page, and you do that by putting the following script tag just before the closing tag in the **app/views/index.ejs** file:

<script type="text/javascript" src="/js/angular/angular.js"></script>

The whole AngularJS application will reside in the **public** folder (resides in the root folder of the application, along with the folders **app** and **config**) of our Express application that we built in the previous tutorials so far, so that every file is available **statically**. Next, in the **public** folder create a file **application.js**which will be our main starting point, and put the following code inside:

var appName = 'mean';

var app = angular.module(appName, **[]**);

angular.element(document).ready(function() {

angular.bootstrap(document, **[**appName**]**);

});

#### Modules

In order to understand what we did in application.js file, you need to know a bit about **modules** in AngularJS: every AngularJS application has **at least one** module. But you can, and should, modularize your code since the goal of [modules](http://en.wikipedia.org/wiki/Modular_programming) is to achieve a [separation of concerns](http://en.wikipedia.org/wiki/Separation_of_concerns) by defining public APIs and limiting the visibility of behavior (functions) and data (properties and variables). In AngularJS the module is defined simply as:

angular.module('mean', **[]**);

First parameter is the name of the module (in our case ***mean***), while the second one is an array of**dependencies** that your module depends on (none in this case, but you'll see it in the following examples). In the HTML document that hosts our application, we can instruct AngularJS to load the application module by passing its name to the **ng-app** directive (more about directives in the next section) like this:

<body ng-app="mean">

<!-- Other awesome code -->

</body>

Great, so we covered first two lines of our code, but what does the rest of it mean? Well, in most cases the first two will do you just fine, but since we'll need a little more leverage later (additional configuration), we used the so called **bootstraping**, and you can learn more about it in the [official docs](https://docs.angularjs.org/guide/bootstrap).

#### Directives

These are either **attributes** or **element** names that actually extend HTML. There are 4 main core directives (***ng-app***,***ng-controller***,***ng-model***,***ng-repeat*** and **ng-show**/**ng-hide**). As many say, custom directives are what make AngularJS so cool, because you can then have something like the following example:

//in the .js file:

angular.module('app')

.directive('alertMessage', function() {

return function(scope, element, attrs) {

element.text(attrs.alertMessage);

};

});

//in the HTML

The message is <h1 alert-message="Howdy"></h1>.

There are actually whole books written on this subject like [this](http://amzn.to/1MtHdn0) or [this](http://amzn.to/1MtHnLi) and we won't go any further with this (nor use it to make our own custom directives smileySad  
), but for starters if you want to get more into all the directive awesomeness here's a good [official tutorial](https://docs.angularjs.org/guide/directive)from Angular team.

#### Folder structure

We spoke already about how folder structure is important when the application gets big, and I've noted how nice it is when you run into a well known environment, so in that case we'll take the word of experienced contributors at MEAN.js and do it [*their way*](https://www.youtube.com/watch?v=LZA36YIu2yU); create a folder named **example** in your **public**folder. In it create a file **example.client.module.js** and paste the following code in it:

angular.module('example', **[]**);

In order to be able to use this module, you need to include the following script tag in the **index.ejs** file, just above the inclusion of the **application.js** file:

<script type="text/javascript" src="/example/example.client.module.

js"></script>

Additionally, now you're going to add your first dependency. Go to the main **application.js** file and make the corresponding line look like this:

var app = angular.module(appName, **[**'example'**]**);

Namely, you've added the **example** module dependency to the main application module. Shortly now, in the [Controllers & $scope](https://hackhands.com/finishing-Angular-TODO-application-deploying-production/#controllers) section I'll explain why is this dependency thingy so cool.

#### Views

Simply put

AngularJS views are HTML templates rendered by the AngularJS compiler to produce a manipulated DOM on your page.

#### Controllers & $scope

You can think of **$scope** as the bond between the **view** and the **controller**. Using a scope object ($scope), the controller can manipulate the model, which in turn then automatically propagates these changes to the view and vice versa. Differently put, scopes provide a [single source of truth](http://en.wikipedia.org/wiki/Single_Source_of_Truth) within your application - no matter where you display some data in your view layer you should only have to change it in one place (a scope property), and the change should automatically propagate throughout the view.

Controller instances are usually created when you use the **ng-controller** directive on an element like this:

<div ng-controller="MyController">

Important to note is that scopes can inherit the model from their parent scope, where the $rootScope is the highest in the "hierarchical chain" of scopes. More on $rootScope [here](https://docs.angularjs.org/guide/scope).

The controllers in AngularJS are defined like in the following example:

angular.module('example').controller('MyController', **[**'$scope',

function($scope) {

$scope.someMessage = "Hi MEAN people! ;)";

}

**]**);

The reason why the above code wasn't written like the following example (although working):

angular.module('example').controller('MyController', function($scope) {

$scope.someMessage = "Hi MEAN people! ;)";

});

is due to the fact that once you use any [**obfuscation**](https://en.wikipedia.org/wiki/Obfuscation_(software)) tools (even simple minimization with variable replacement option) your code will stop working in the second example because dependencies will get mixed up, and the code in the first example solves this. You will learn more about scope and controllers as we go on, but if you're interested feel free to take a detour now and learn more about [scope](https://docs.angularjs.org/guide/scope) and [controllers](https://docs.angularjs.org/guide/controller)from the official docs (both quite extensive and good).

#### Dependency injection

Now, this fancy term can be nicely explained with a simple example; say you have a simple **Howdy** class which creates an instance of **SomeService**, and when the **greet** function is called it checks for it's value and if it's equal to 'iLoveToReturnUsefulData' you can do something great further within your app logic:

var Howdy = function() {

this.someService = new SomeService();

};

Hello.prototype.greet = function() {

var data = this.someService.getData();

if (data.value === 'iLoveToReturnUsefulData')

//do something great

};

And sure, this works well until you want to test shocked  
this class. You'll create an instance of the **Howdy** class in your test function, but you won't be able to pass**someService** object to it. Dependency injection solves this by moving the responsibility of creating the someService object to the creator of the Howdy instance, whether it is another object or a [test mock object](http://en.wikipedia.org/wiki/Mock_object):

var Howdy = function(someService) {

this.someService = someService;

};

With this rather small change you're able to control the behavior of the Howdy class instance **outside of it's constructor**, which is often referred to as [inversion of control](http://en.wikipedia.org/wiki/Inversion_of_control). You can learn more about DI in the[official docs](https://docs.angularjs.org/guide/di). One of the good explanations about how dependency injection works in AngularJS I found was:

You need a resource from Angular, the container. To get it, you write a function that declares a parameter with > exactly> the same name as the resource. The container detects your function, matches the parameter with the resource, makes an instance of the resource and calls your function with this instance as the argument for your parameter. Since the container has the active role (as opposed to you actively requesting the instance of the resource in a statement), dependency injection is considered an [inversion of control](http://martinfowler.com/bliki/InversionOfControl.html), colorfully described as an application of the [Hollywood principle](http://en.wikipedia.org/wiki/Hollywood_principle), "Don't call us, we'll call you."

#### Routing with ngRouter

The [ngRoute](https://docs.angularjs.org/api/ngRoute) module allows you to define URL paths and their corresponding templates, which will be rendered whenever the user navigates to those paths. You may stumble upon [ui-router](https://github.com/angular-ui/ui-router), which is an advanced version for routing and it's actually a third party library. You can learn more about the differences from [this StackOverflow question](http://stackoverflow.com/questions/21023763/difference-between-angular-route-and-angular-ui-router).

Since ngRoute is not a part of the core AngularJS modules, you need to install it with bower by executing the following command (from the root directory of the project):

bower install angular-route

Next, you need to add this script file in **app/views/index.ejs** just after the angular inclusion script:

<script type="text/javascript" src="/js/angular-route/angular-route.min.js"></script>

Also, you have to add the ngRoute dependency in your main application.js file:

var app = angular.module(appName, **[**'ngRoute', 'example'**]**);

To mark your application routes as single-page application routes, you will need to use a routing scheme called Hashbangs which are implemented by adding an exclamation mark right after the hash sign, so an example URL would be <http://localhost:1337/#!/example>. You can learn more about hashbangs [here](https://docs.angularjs.org/guide/$location), and in order to configure your application routing go to the **public/application.js** file and append the following code:

app.config(**[**'$locationProvider', function($locationProvider) {

$locationProvider.hashPrefix('!');

}

**]**);

#### Build me some routes

In the **public/example** folder create a folder named **config** and inside it a file **example.client.routes.js** with the following content:

angular.module('example').config(**[**'$routeProvider',

function($routeProvider) {

$routeProvider.

when('/', {

templateUrl: 'example/views/example.client.view.html'

}).

otherwise({

redirectTo: '/'

});

}

**]**);

The previous mumbo jumbo explained in quick rundown:

* with angular.module('example') method you fetched the **example** module and
* executed the **config()** method in which you applied the aforementioned dependency injection to inject the **$routeProvider** (available from ngRoute module),
* where you used his **when()** method to define a new route along with the route URL and the template URL
* **otherwise()** method defines the behavior of the router when the user navigates to an undefined URL

ngRoute module also comes with the **ng-view** directive which tells the AngularJS router which DOM element to use to render the routing views. In order to use this, add the following line to your**app/views/index.ejs** file:

<!--add this just after <body> tag-->

<section ng-view></section>

<!--add this after the controllers/example.client.controller.js script tag-->

<script type="text/javascript" src="/example/config/example.client.routes.js"></script>

#### Services

Basically, you'll use AngularJS services to fetch data from your server, or share the data between controllers. Some services ([$http](https://docs.angularjs.org/api/ng/service/$http), [$resource](https://docs.angularjs.org/api/ngResource/service/$resource), [$location](https://docs.angularjs.org/api/ng/service/$location), ...) come pre-built within the core part of AngularJS, while you can also create your own with [factory](https://docs.angularjs.org/guide/providers) or [service](https://docs.angularjs.org/guide/providers)(yeah, I know, I know a service service - ng-WAT moment for those who saw the Reznik's video). Just for the sake of an example, a simple service may look like this (accompanying [jsFiddle example here](http://jsfiddle.net/Hitman666/whx4vs0y/1/)):

angular.module('example').service('ExampleService', **[**

function() {

this.someValue = 'MEAN Application';

this.someMethod = function() {}

}

**]**);

which you would then inject and use in some controller like this:

angular.module('example').controller('ExampleController', **[**'$scope', 'ExampleService',

function($scope, ExampleService) {

$scope.name = ExampleService.someValue;

}

**]**);

### Handling user authentication the mean way

Express application will render the user object directly in the EJS view and then use an AngularJS service to wrap that object into its tender hands. So, let's get cracking; in the file**app/controllers/index.server.controller.js**alter the user definition to look like this:

user: JSON.stringify(req.user)

Next, in the **app/views/index.ejs** file add the following code:

<script type="text/javascript">

window.user = <%- user || 'null' %>;

</script>

which will in turn render the user object as a JSON representation right in your main view application. Once the AngularJS application bootstrap is finished, the authentication state will already be available.

#### Authentication service

Create a new folder named **users** in the **public** folder, and in it create a file named**users.client.module.js**with the following content:

angular.module('users', **[]**);

Note for the avid readers that are following with the cloned project from Github, the folder is actually called useri but it doesn't mind if you rename to users, just make sure you rename the script inclusion paths also in that case.Now, inside the **users** folder create a folder called **services** and inside it create a file**authentication.client.service.js**with the following content:

angular.module('users').factory('Authentication', **[**

function() {

this.user = window.user;

return {

user: this.user

};

}

**]**);

With this you referenced the **window.user** object from the AngularJS service. Naturally, you still need to reference these two new files in your main **views/index.ejs** file (add just above the application.js inclusion script tag):

<script type="text/javascript" src="/users/users.client.module.

js"></script>

<script type="text/javascript" src="/users/services/authentication.

client.service.js"></script>

Hopeful reader as you are, you know by now that you have to inject the new users module in the main application.js file:

var app = angular.module(appName, **[**'ngRoute', 'example', 'users'**]**);

#### Facebook redirect bug

Additionally, in order to solve the [Facebook redirect bug](https://github.com/amoshaviv/mean-web-development/pull/2/files?diff=split) that adds a hash part to the application's URL after the OAuth authentication round-trip add the following line to application.js file:

if (window.location.hash === '#\_=\_') window.location.hash = '#!';

Finally, you can use the Authentication service by injecting the Authentication service in the**example/controllers/example.client.controller.js** file:

angular.module('example').controller('ExampleController', **[**'$scope', 'Authentication',

function($scope, Authentication) {

$scope.authentication = Authentication;

}

**]**);

## TODO CRUD elements

In this part you'll first learn about the changes that you need to make to the Express part, and then in the AngularJS part of your application. So let's start with Express to move a bit from AngularJS :)

### Express CRUDe elements of our ever so slightly MEAN TODO app

P.S. (wait, what!?, a postscript in the middle of an article, are you for real? - dear readers don't worry, we're past the 5k word tutorial so I'm slightly starting to loose it ;). Oh, but professionalism and all that?! - it ain't work if its fun, right? ;)) For those who kind of felt this "ever so slightly" sounded familiar I'm saying: 'Yeah, I too can't wait for[*The Doors of Stone*](http://www.goodreads.com/book/show/21032488-doors-of-stone)'.

Ok, FFS, enough with the theory back 'n forthing I want to see some codez plz. Ok, ok, I hear you; what were going to do now is implement the CRUD parts of our TODO application. However, since I showed you in baby steps the few mains parts of AngularJS, now we're going to tackle this like big boys/gals and I'll give you the exact full code which you have to put in each of the files and then I'll comment their crucial parts.

#### Model me nicely

In the **app/models** folder of your Express.js app make a new file called **todo.server.model.js** with the following full content that will be used for ever and ever:

var mongoose = require('mongoose'),

Schema = mongoose.Schema;

var TodoSchema = new Schema({

created: {

type: Date,

default: Date.now

},

title: {

type: String,

default: '',

trim: true,

required: "Title can't be blank"

},

comment: {

type: String,

default: '',

trim: true

},

creator: {

type: Schema.ObjectId,

ref: 'User'

},

completed: {

type: Boolean,

default: false

}

});

mongoose.model('Todo', TodoSchema);

If this looks foreign, then I beg you please go and brush up on Mongoose in our third tutorial which was all about [MongoDB, Mongoose and Passport Authentication](https://hackhands.com/mongodb-crud-mvc-way-with-passport-authentication/).

In this code you created a Mongoose Todo model with five model fields (created, title, comment, creator, completed) whose meaning should speak for itself. An option worth mentioning additionally is that we made a reference to the User model object for the creator field so that we'll have the info user actually created the certain todo.

Next, in the **config/mongoose.js** file and add the following line:

require('../app/models/todo.server.model');

Just for your reference, the whole contents of the file should look like this:

var config = require('./config'),

mongoose = require('mongoose');

module.exports = function() {

var db = mongoose.connect(config.db);

require('../app/models/user.server.model');

require('../app/models/todo.server.model');

return db;

};

#### Control all the things

Now you're going to set up the controller for your CRUD part of the TODO thingies. In the **app/controllers**folder create a new file **todos.server.controller.js** with the full following content:

var mongoose = require('mongoose'),

Todo = mongoose.model('Todo');

var getErrorMessage = function(err) {

if (err.errors) {

for (var errName in err.errors) {

if (err.errors**[**errName**]**.message) return err.errors**[**errName**]**.message;

}

} else {

return 'Unknown server error';

}

};

exports.create = function(req, res) {

var todo = new Todo(req.body);

todo.creator = req.user;

todo.save(function(err) {

if (err) {

return res.status(400).send({

message: getErrorMessage(err)

});

} else {

res.json(todo);

}

});

};

exports.list = function(req, res) {

Todo.find().sort('-created').populate('creator', 'name username').exec(function(err, todos) {

if (err) {

return res.status(400).send({

message: getErrorMessage(err)

});

} else {

res.json(todos);

}

});

};

exports.read = function(req, res) {

res.json(req.todo);

};

exports.todoByID = function(req, res, next, id) {

Todo.findById(id).populate('creator', 'name username').exec(function(err, todo) {

if (err)

return next(err);

if (!todo)

return next(new Error('Failed to load todo ' + id));

req.todo = todo;

next();

});

};

exports.update = function(req, res) {

var todo = req.todo;

todo.title = req.body.title;

todo.comment = req.body.comment;

todo.completed = req.body.completed;

todo.save(function(err) {

if (err) {

return res.status(400).send({

message: getErrorMessage(err)

});

} else {

res.json(todo);

}

});

};

exports.delete = function(req, res) {

var todo = req.todo;

todo.remove(function(err) {

if (err) {

return res.status(400).send({

message: getErrorMessage(err)

});

} else {

res.json(todo);

}

});

};

exports.hasAuthorization = function(req, res, next) {

if (req.todo.creator.id !== req.user.id) {

return res.status(403).send({

message: 'User is not authorized'

});

}

next();

};

This, rather nicely structured, mumbo jumbo explained per functions:

* **getErrorMessage** - iterates over the Mongoose errors and returns the first one
* **create**
  + creates a new Todo model instance using the HTTP request body
  + adds the authenticated Passport user as the todo creator
  + saves the ***todo*** document using Mongoose instance **save** method
* **list**
  + fetches the collection of **todo** documents by using the Mongoose instance **find** method
  + sorts them by **created** property
  + adds **name** and ***username***\_fields to to the \*\*\_creator\*\* property
* **read** - outputs the todo object as JSON
* **todoById**- fetches the ***todo*** document by using the Mongoose instance **findById** method.
* **update** - updates the ***todo*** document which you've previously fetched with the todoById middleware (if you remeber middlewares that is, if not, please go and refresh your memory in the second tutorial which was all about [Node.js and Express](https://hackhands.com/delving-node-js-express-web-framework/))
* **delete** - deletes the ***todo*** document which you've previously fetched with the todoById middleware
* **hasAuthorization** - middleware that uses the **req.todo** and **req.user** objects to verify that the current user is the creator of the current todo.

#### To continue you need to say the magic word

You need to know which user is authenticated so that you know which one made a certain action (like for example created a new todo). You can check this inside every method, but that's just code duplication since you can use the Express middleware chaining to block unauthorized requests from executing your controller methods.

So, in the **app/controllers/users.server.controller.js** file add the following code:

exports.requiresLogin = function(req, res, next) {

if (!req.isAuthenticated()) {

return res.status(401).send({

message: 'User is not logged in'

});

}

next();

};

This middleware uses the Passport initiated **req.isAuthenticated** method to check whether a user is currently authenticated. If it finds out the user is indeed signed in, it will call the next middleware in the chain; otherwise it will respond with an authentication error and an HTTP error code. Full code listing for this file is below:

var User = require('mongoose').model('User'),

passport = require('passport');

var getErrorMessage = function(err) {

var message = '';

if (err.code) {

switch (err.code) {

case 11000:

case 11001:

message = 'Username already exists';

break;

default:

message = 'Something went wrong';

}

} else {

for (var errName in err.errors) {

if (err.errors**[**errName**]**.message)

message = err.errors**[**errName**]**.message;

}

}

return message;

};

exports.renderLogin = function(req, res, next) {

if (!req.user) {

res.render('login', {

title: 'Log-in Form',

messages: req.flash('error') || req.flash('info')

});

} else {

return res.redirect('/');

}

};

exports.renderRegister = function(req, res, next) {

if (!req.user) {

res.render('register', {

title: 'Register Form',

messages: req.flash('error')

});

} else {

return res.redirect('/');

}

};

exports.register = function(req, res, next) {

if (!req.user) {

var user = new User(req.body);

var message = null;

user.provider = 'local';

user.save(function(err) {

if (err) {

var message = getErrorMessage(err);

req.flash('error', message);

return res.redirect('/register');

}

req.login(user, function(err) {

if (err)

return next(err);

return res.redirect('/');

});

});

} else {

return res.redirect('/');

}

};

exports.logout = function(req, res) {

req.logout();

res.redirect('/');

};

exports.saveOAuthUserProfile = function(req, profile, done) {

User.findOne({

provider: profile.provider,

providerId: profile.providerId

},

function(err, user) {

if (err) {

return done(err);

} else {

if (!user) {

var possibleUsername = profile.username || ((profile.email) ? profile.email.split('@')**[**0**]** : '');

User.findUniqueUsername(possibleUsername, null, function(availableUsername) {

profile.username = availableUsername;

user = new User(profile);

user.save(function(err) {

if (err) {

var message = \_this.getErrorMessage(err);

req.flash('error', message);

return res.redirect('/register');

}

return done(err, user);

});

});

} else {

return done(err, user);

}

}

}

);

};

exports.create = function(req, res, next) {

var user = new User(req.body);

user.save(function(err) {

if (err) {

return next(err);

} else {

res.json(user);

}

});

};

exports.list = function(req, res, next) {

User.find({}, function(err, users) {

if (err) {

return next(err);

} else {

res.json(users);

}

});

};

exports.read = function(req, res) {

res.json(req.user);

};

exports.userByID = function(req, res, next, id) {

User.findOne({

\_id: id

},

function(err, user) {

if (err) {

return next(err);

} else {

req.user = user;

next();

}

}

);

};

exports.update = function(req, res, next) {

User.findByIdAndUpdate(req.user.id, req.body, function(err, user) {

if (err) {

return next(err);

} else {

res.json(user);

}

});

};

exports.delete = function(req, res, next) {

req.user.remove(function(err) {

if (err) {

return next(err);

} else {

res.json(req.user);

}

})

};

exports.requiresLogin = function(req, res, next) {

if (!req.isAuthenticated()) {

return res.status(401).send({

message: 'User is not logged in'

});

}

next();

};

#### Routes coming up, Expressly

In the **app/routes** folder create a new file **todos.server.routes.js**:

var users = require('../../app/controllers/users.server.controller'),

todos = require('../../app/controllers/todos.server.controller');

module.exports = function(app) {

app.route('/api/todos')

.get(todos.list)

.post(users.requiresLogin, todos.create);

app.route('/api/todos/:todoId')

.get(todos.read)

.put(users.requiresLogin, todos.hasAuthorization, todos.update)

.delete(users.requiresLogin, todos.hasAuthorization, todos.delete);

app.param('todoId', todos.todoByID);

};

These route definitions should be familiar to you, but just as a special note you should notice how the **post**method uses the **users.requiresLogin** middleware, and how **put** and **delete** methods use that and also**todos.hasAuthorization** middleware to make sure that authenticated users create new todos and that only user who is the actual creator of the certain todo can delete or update it.

The **app.param** method makes sure every route that has the **todoId** parameter will first call the**todos.todoByID**middleware.

Finally, in the **config/express.js** file you need to include this new route:

require('../app/routes/todos.server.routes.js')(app);

As a reference, the whole contents of this file should be as follows:

var config = require('./config'),

express = require('express'),

bodyParser = require('body-parser'),

passport = require('passport'),

flash = require('connect-flash'),

session = require('express-session');

module.exports = function() {

var app = express();

app.use(bodyParser.urlencoded({

extended: true

}));

app.use(bodyParser.json());

app.use(session({

saveUninitialized: true,

resave: true,

secret: 'OurSuperSecretCookieSecret'

}));

app.set('views', './app/views');

app.set('view engine', 'ejs');

app.use(flash());

app.use(passport.initialize());

app.use(passport.session());

require('../app/routes/index.server.routes.js')(app);

require('../app/routes/users.server.routes.js')(app);

require('../app/routes/todos.server.routes.js')(app);

app.use(express.static('./public'));

return app;

};

### Angular CRUDe elements of our ever so slightly MEAN TODO app

Wait, didn't I see this title before?! Yeah, kind of, but this one has Angular in it ;), and no Patrick Rothfuss reference smileySad

The following few steps should feel very familiar (same [M.O.](https://en.wikipedia.org/wiki/Modus_operandi) as when we were adding the [ngRoute module](https://hackhands.com/finishing-Angular-TODO-application-deploying-production/#ngrouter)). Since we designed our Express API in a [RESTful](https://en.wikipedia.org/wiki/Representational_state_transfer) manner, we can make use of the external[ngResource](https://docs.angularjs.org/api/ngResource) module. First, install ngResource with bower:

bower install angular-resource

Next, you need to add this script file in **app/views/index.ejs** just after the ngRoute inclusion script:

<script type="text/javascript" src="/js/angular-resource/angular-resource.min.js"></script>

Also, you have to add the ngResource dependency in your main application.js file:

var app = angular.module(appName, **[**'ngResource', 'ngRoute', 'example', 'users'**]**);

#### MVC structure

In the **public** folder create a new folder **todos** and inside it create a new file **todos.client.module.js** with the following content:

angular.module('todos', **[]**);

Also, inside **todos** folder create four new folders: **controllers**, **views**, **services** and **config**. Now, you have to add the **todos** module dependency in your main **application.js** file:

var app = angular.module(appName, **[**'ngResource', 'ngRoute', 'example', 'users', 'todos'**]**);

Since we won't be messing with **application.js** file anymore, here's how it should look like for your reference:

var appName = 'mean';

var app = angular.module(appName, **[**'ngResource', 'ngRoute', 'example', 'users', 'todos'**]**);

app.config(**[**'$locationProvider', function($locationProvider) {

$locationProvider.hashPrefix('!');

}

**]**);

if (window.location.hash === '#\_=\_') window.location.hash = '#!';

angular.element(document).ready(function() {

angular.bootstrap(document, **[**appName**]**);

});

#### Service

Create a new file **todos.client.service.js** file in the **public/todos/services** folder with the following content:

angular.module('todos').factory('Todos', **[**'$resource',

function($resource) {

return $resource('api/todos/:todoId', {

todoId: '@\_id'

}, {

update: {

method: 'PUT'

}

});

}

**]**);

The **Todos** service that you just created uses the **$resource** factory which has three parameters:

* base URL for the resource endpoint
* a routing parameter assignment using the todo's document **\_id** field, and
* actions argument extending the resource methods with an **update** method that uses the PUT HTTP method.

#### Controller

Create a new file **todos.client.controller.js** file in the **public/todos/controllers**folder with the following content:

angular.module('todos').controller('TodosController', **[**'$scope', '$routeParams', '$location', 'Authentication', 'Todos',

function($scope, $routeParams, $location, Authentication, Todos) {

$scope.authentication = Authentication;

$scope.create = function() {

var todo = new Todos({

title: this.title,

comment: this.comment

});

todo.$save(function(response) {

$location.path('todos/' + response.\_id);

}, function(errorResponse) {

$scope.error = errorResponse.data.message;

});

};

$scope.find = function() {

$scope.todos = Todos.query();

};

$scope.findOne = function() {

$scope.todo = Todos.get({

todoId: $routeParams.todoId

});

};

$scope.update = function() {

$scope.todo.$update(function() {

$location.path('todos/' + $scope.todo.\_id);

}, function(errorResponse) {

$scope.error = errorResponse.data.message;

});

};

$scope.delete = function(todo) {

if (todo) {

todo.$remove(function() {

for (var i in $scope.todos) {

if ($scope.todos**[**i**]** === todo) {

$scope.todos.splice(i, 1);

}

}

});

} else {

$scope.todo.$remove(function() {

$location.path('todos');

});

}

};

}

**]**);

In this mumbo jumbo we're injecting four services ($routeParams, $location, Authentication and Todos), and we have various functions defined on the scope object (which basically do what their names suggest). A note on the **create** function is that **title** and **comment** field will be provided via the form that we'll create shortly now, in the [Views](https://hackhands.com/finishing-Angular-TODO-application-deploying-production/#views)section. Also, the **delete** function figures out whether the user is deleting an todo from a list or directly from the todo view.

#### Views

##### Create a todo view

In the **public/todos/views** folder create a file **create-todo.client.view.html** with the following content:

<div data-ng-controller="TodosController">

<nav class="navbar navbar-default navbar-fixed-top">

<div class="container">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header page-scroll">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#page-top">MEAN TODO</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav navbar-right">

<li class="hidden">

<a href="#page-top"></a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/login">Login</a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/register">Register</a>

</li>

<!--<li class="page-scroll">

<a href="https://github.com/Hitman666/MEAN\_MVC\_4thTutorial">Fork on Github</a>

</li>-->

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/#!/todos">TODOS</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/logout">Hello <span data-ng-bind="authentication.user.name" data-ng-show="authentication.user"></span>, Logout?</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container-fluid -->

</nav>

<section class="container">

<h1>New Todo</h1>

<form class="form-signin" data-ng-submit="create()" novalidate>

<div>

<label for="title" class="sr-only">Title</label>

<div>

<input type="text" data-ng-model="title" id="title" placeholder="Title" required autofocus class="form-control">

</div>

</div>

<div>

<label for="comment" class="sr-only">Comment</label>

<div>

<textarea data-ng-model="comment" id="comment" cols="30" rows="10" placeholder="Comment" class="form-control"></textarea>

</div>

</div>

<div>

<button type="submit" class="btn btn-lg btn-primary btn-block">Create TODO</button>

</div>

<div data-ng-show="error">

<strong data-ng-bind="error"></strong>

</div>

</form>

</section>

</div>

If you look past the styling, the navigation all other designery stuff, you'll see that what's important here is the form with the **title** input and the comment **input**. Important to note is that the form has a following attribute:

data-ng-submit="create()"

and this is AngularJS was of saying "Yo, when the users clicks the submit button, let the***create***function handle it". The same **create** function you defined in the Controllers section. Btw, don't be confused with the term function vs method. They're basically the same, it's just that one is from Java world and other from PHP (well, at least in my case :P). In this file you can also see the **ng-show** and **ng-hide** directives in action.

##### View a todo view

In the **public/todos/views** folder create a file **view-todo.client.view.html** with the following content:

<div data-ng-controller="TodosController">

<nav class="navbar navbar-default navbar-fixed-top">

<div class="container">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header page-scroll">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#page-top">MEAN TODO</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav navbar-right">

<li class="hidden">

<a href="#page-top"></a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/login">Login</a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/register">Register</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/#!/todos/create">CREATE</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/#!/todos">TODOs</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/logout">Hello <span data-ng-bind="authentication.user.name" data-ng-show="authentication.user"></span>, Logout?</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container-fluid -->

</nav>

<section data-ng-init="findOne()" class="container">

<h1 data-ng-bind="todo.title" ng-class="todo.completed?'completed':''"></h1>

<div data-ng-show="authentication.user.\_id == todo.creator.\_id">

<a href="/#!/todos/{{todo.\_id}}/edit">edit</a>

<a href="#" data-ng-click="delete();">delete</a>

</div>

<small>

<em data-ng-bind="todo.created | date:'mediumDate'"></em>

</small>

<p data-ng-bind="todo.comment"></p>

</section>

</div>

Again, the important part here is the one in the <section> tag where you show the todo title, creation date and comment. Additionally, if the logged in user is the creator of the particular todo then two additional links will be shown (edit and delete) using the ng-show directive. The **data-ng-bind** is actually the same as if you would write **ng-bind**, but some validators tend to complain for the latter (you can learn more about it [here](http://stackoverflow.com/questions/16589853/ng-app-vs-data-ng-app-what-is-the-difference)). Also, if you ever wondered what's the deal with ng-bind vs ng-model, you can learn more [here](http://stackoverflow.com/questions/12419619/whats-the-difference-between-ng-model-and-ng-bind), but a very nice summary is this:

* **ng-bind** has one-way data binding ($scope -> view). It has a shortcut {{val}} which displays the scope value $scope.val inserted into HTML where val is a variable name.
* **ng-model** is intended to be put inside of form elements and has two-way data binding ($scope -> view and view -> $scope) e.g. input ng-model="val".

There's one more novelty in this file - the **data-ng-init** directive. Yes, you've guessed it right from the name - it actually triggers the function **findOne**upon initialization, defined before on the scope object of**TodosController** controller.

##### Edit a todo view

In the **public/todos/views** folder create a file **edit-todo.client.view.html** with the following content:

<div data-ng-controller="TodosController">

<nav class="navbar navbar-default navbar-fixed-top">

<div class="container">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header page-scroll">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#page-top">MEAN TODO</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav navbar-right">

<li class="hidden">

<a href="#page-top"></a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/login">Login</a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/register">Register</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/#!/todos/create">CREATE</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/#!/todos">TODOs</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/logout">Hello <span data-ng-bind="authentication.user.name" data-ng-show="authentication.user"></span>, Logout?</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container-fluid -->

</nav>

<section data-ng-init="findOne()" class="container">

<h1>Edit Todo</h1>

<form data-ng-submit="update()" novalidate class="form-signin">

<div>

<label for="title" class="sr-only">Title</label>

<div>

<input type="text" data-ng-model="todo.title" id="title" placeholder="Title" required autofocus class="form-control">

</div>

</div>

<div>

<div>

Completed: <input type="checkbox" data-ng-model="todo.completed" ng-checked="todo.completed" name="completed" id="completed" class="form-control" />

</div>

</div>

<div>

<label for="comment" class="sr-only">Comment</label>

<div>

<textarea data-ng-model="todo.comment" id="comment" cols="30" rows="10" placeholder="Comment" class="form-control"></textarea>

</div>

</div>

<div>

<button type="submit" class="btn btn-lg btn-primary btn-block">Update TODO</button>

</div>

<div data-ng-show="error">

<strong data-ng-bind="error"></strong>

</div>

</form>

</section>

</div>

Here also is the part in the <section> tag where the party at, where you're able to edit the **title**, **comment**and **completion** of the todo. AngularJS handles clicking the submit button in the **update** function, and you know this by the following attribute in the form tag:

data-ng-submit="update()"

##### List todos view

In the **public/todos/views** folder create a file **list-todos.client.view.html** with the following content:

<div data-ng-controller="TodosController">

<nav class="navbar navbar-default navbar-fixed-top">

<div class="container">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header page-scroll">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#page-top">MEAN TODO</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav navbar-right">

<li class="hidden">

<a href="#page-top"></a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/login">Login</a>

</li>

<li class="page-scroll" data-ng-show="!authentication.user">

<a href="/register">Register</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/#!/todos/create">CREATE</a>

</li>

<li class="page-scroll" data-ng-show="authentication.user">

<a href="/logout">Hello <span data-ng-bind="authentication.user.name" data-ng-show="authentication.user"></span>, Logout?</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container-fluid -->

</nav>

<section data-ng-init="find()">

<div class="container">

<h1>Todos</h1>

Show completed: <input type="checkbox" ng-model="showcompleted" ng-init="showcompleted=true">

<ul>

<li data-ng-repeat="todo in todos" ng-if="showcompleted && todo.completed || !todo.completed">

<a data-ng-href="#!/todos/{{todo.\_id}}" data-ng-bind="todo.title" ng-class="todo.completed?'completed':''"></a>

(<small data-ng-bind="todo.created | date:'short'"></small>)

<br>

<small data-ng-bind="todo.comment" class="comment"></small>

</li>

</ul>

<div data-ng-hide="!todos || todos.length">

No todos yet, why don't you <a href="/#!/todos/create">create one</a>?

</div>

</div>

</section>

</div>

Here also, yada yada yada (you're seeing a pattern here, ait?), the part in the <section> tag is most important, where you're listing all the todos. Additionally, with the use of a checkbox with a**showcompleted**model:

Show completed: <input type="checkbox" ng-model="showcompleted" ng-init="showcompleted=true">

you're able to toggle the visibility of the completed todos, by leveraging the **ng-if** directive:

<li data-ng-repeat="todo in todos" ng-if="showcompleted && todo.completed || !todo.completed">

#### ****Show me the routes****

In the **public/todos/config** folder create a file **todos.client.routes.js** with the following content:

angular.module('todos').config(**[**'$routeProvider',

function($routeProvider) {

$routeProvider.

when('/todos', {

templateUrl: 'todos/views/list-todos.client.view.html'

}).

when('/todos/create', {

templateUrl: 'todos/views/create-todo.client.view.html'

}).

when('/todos/:todoId', {

templateUrl: 'todos/views/view-todo.client.view.html'

}).

when('/todos/:todoId/edit', {

templateUrl: 'todos/views/edit-todo.client.view.html'

});

}

**]**);

As you can see, we're using the $routeProvider service (from ngRoute module) to bind each view with its route. Additionally, worth mentioning are the last two views, which handle an existing **todo**and include the**todoId** route parameters in their URL definition. This will enable your controller to extract the**todoId**parameter using the **$routeParams** service.

#### ****Stitch me up and send back to battlefield****

As a final thing, you have to include all these AngularJS files that we created in the main **index.ejs** view. For your reference, the whole **/app/views/index.ejs** file should look like this:

<!DOCTYPE html>

<html xmlns:ng="http://angularjs.org">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<meta name="author" content="Nikola Brežnjak">

<title>MEAN TODO</title>

<link href="css/bootstrap.css" rel="stylesheet">

<link href="css/mean.css" rel="stylesheet">

<!-- Custom Fonts -->

<link href="css/font-awesome/css/font-awesome.min.css" rel="stylesheet" type="text/css">

<link href="http://fonts.googleapis.com/css?family=Montserrat:400,700" rel="stylesheet" type="text/css">

<link href="http://fonts.googleapis.com/css?family=Lato:400,700,400italic,700italic" rel="stylesheet" type="text/css">

</head>

<body ng-app="mean">

<section ng-view class="main-section"></section>

<script type="text/javascript">

window.user = <%- user || 'null' %>;

</script>

<script type="text/javascript" src="/js/jquery.js"></script>

<script type="text/javascript" src="/js/bootstrap.min.js"></script>

<script type="text/javascript" src="/js/mean.js"></script>

<script type="text/javascript" src="/js/angular/angular.js"></script>

<script type="text/javascript" src="/js/angular-route/angular-route.min.js"></script>

<script type="text/javascript" src="/js/angular-resource/angular-resource.min.js"></script>

<script type="text/javascript" src="/todos/todos.client.module.js"></script>

<script type="text/javascript" src="/todos/controllers/todos.client.controller.js"></script>

<script type="text/javascript" src="/todos/services/todos.client.service.js"></script>

<script type="text/javascript" src="/todos/config/todos.client.routes.js"></script>

<script type="text/javascript" src="/useri/users.client.module.js"></script>

<script type="text/javascript" src="/useri/services/authentication.client.service.js"></script>

<script type="text/javascript" src="/example/example.client.module.js"></script>

<script type="text/javascript" src="/example/controllers/example.client.controller.js"></script>

<script type="text/javascript" src="/example/config/example.client.routes.js"></script>

<script type="text/javascript" src="/application.js"></script>

</body>

</html>

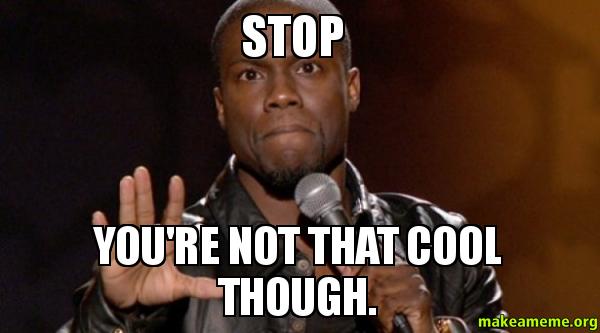
At this point you can run your server (when in the root directory of the project) with:

node server

and have your own meantodo app running locally on port 1337 (make sure you have your mongod running on the standard 27017 port), but there is a better way which we'll cover in the next section.

## Deploying me gently

You've followed the tutorial through to here, you've got everything working locally but



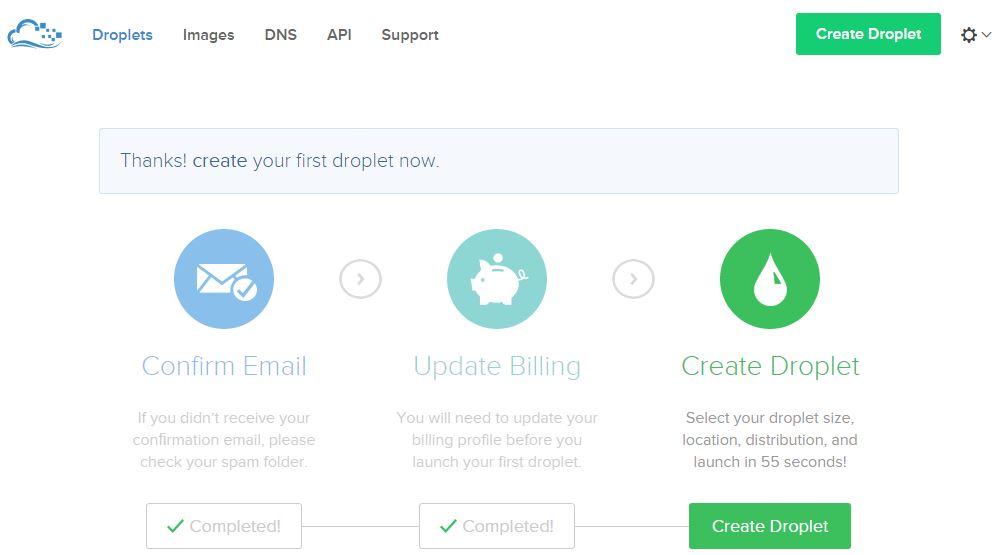
But why? Well, you still need to take this thing you've built to the Internetz (no, it's not a typo, but you know that by now, right?smileyGlasses  
) and show it to the world. Following are the steps to make this happen.

### Getting a server

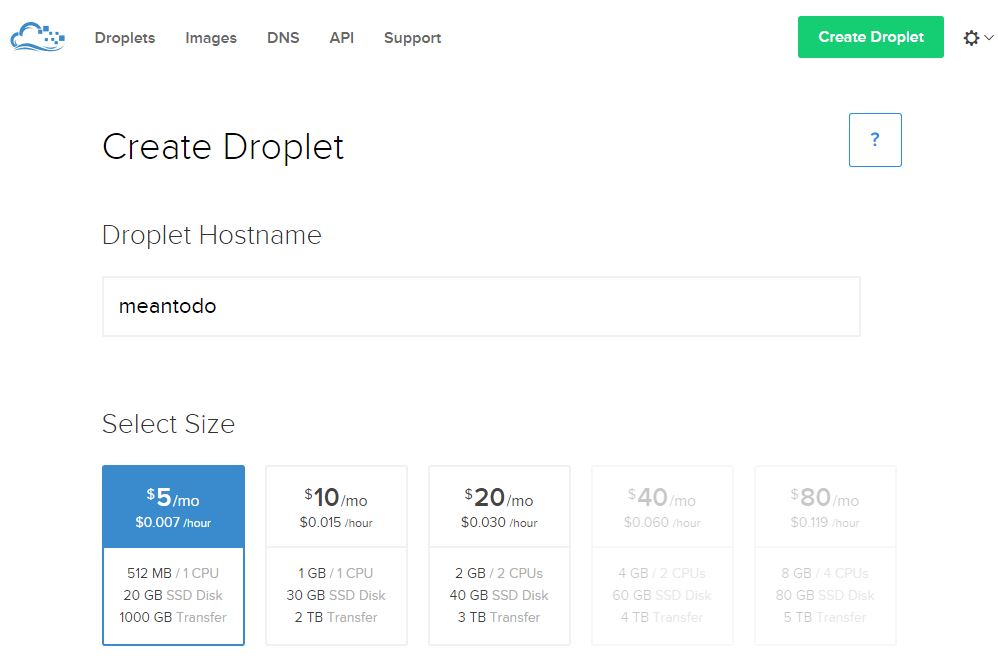
Over the past year I've tried lots of options for hosting my MEAN applications, and finally I've settled for[DigitalOcean](https://www.digitalocean.com/?refcode=974c9bc93d77). It's true what they advertise - that you can have a SSD cloud server up and running in less than 55 seconds. However, you don't have to go spending your hard earned cash just yet - I have a special deal with [DigitalOcean](https://www.digitalocean.com/?refcode=974c9bc93d77) and you can signup via [this link](https://www.digitalocean.com/?refcode=974c9bc93d77) and get 10$ immediately, which will be enough for 2 full months of non stop server running. They even provide you with API so that you can control your server, and [this tutorial](http://code.tutsplus.com/tutorials/using-the-digital-ocean-api-to-manage-cloud-instances--cms-22864) shows you how.

I don't want to make it look like I'm pushing you on DigitalOcean - you can sign up with someone else if you like, I just recommend them because I had a good experience with them in the past and because it's so easy to get a MEAN stack all pre-installed (more on this in a bit) and ready for use.

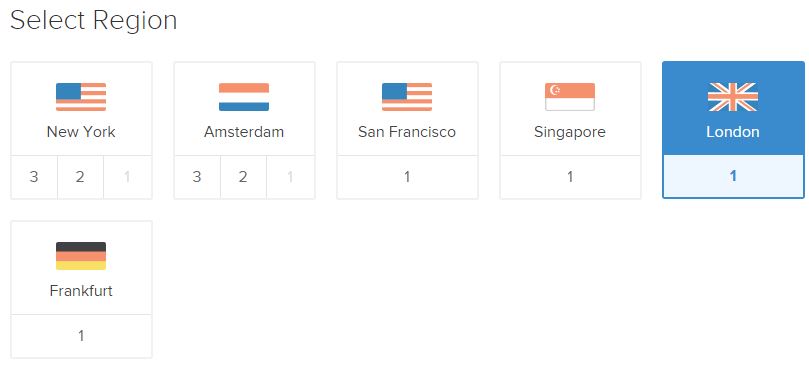
#### Create a droplet

Once you complete the sing up on [DigitalOcean](https://www.digitalocean.com/?refcode=974c9bc93d77) you will be presented with a similar screen, as show on the image below, where you have to click on the **Create Droplet** button:  


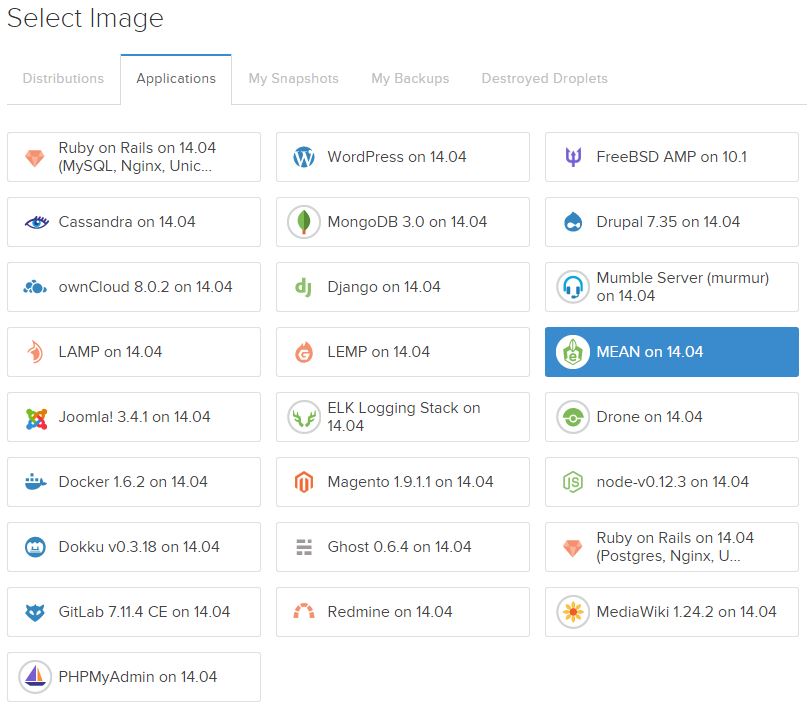
Next, you have to name your **droplet** (cool name the folks at DigitalOcean chose to call their server instances) and select the size (the smallest one will be just fine here), just as it's shown on the image below:



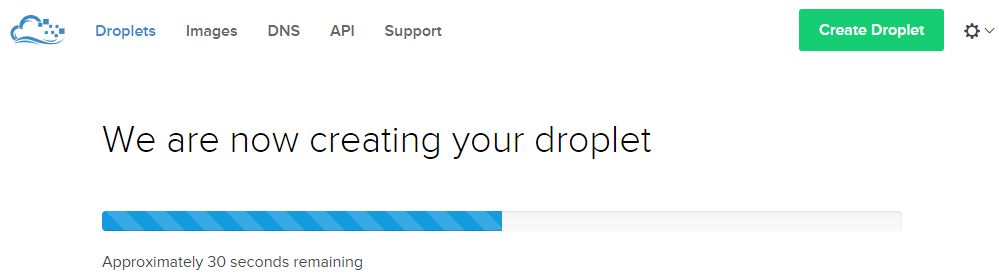
After this, you have to choose a region, as shown on the image below. Naturally, you'll want to choose some location which is closer to your targeted audience.



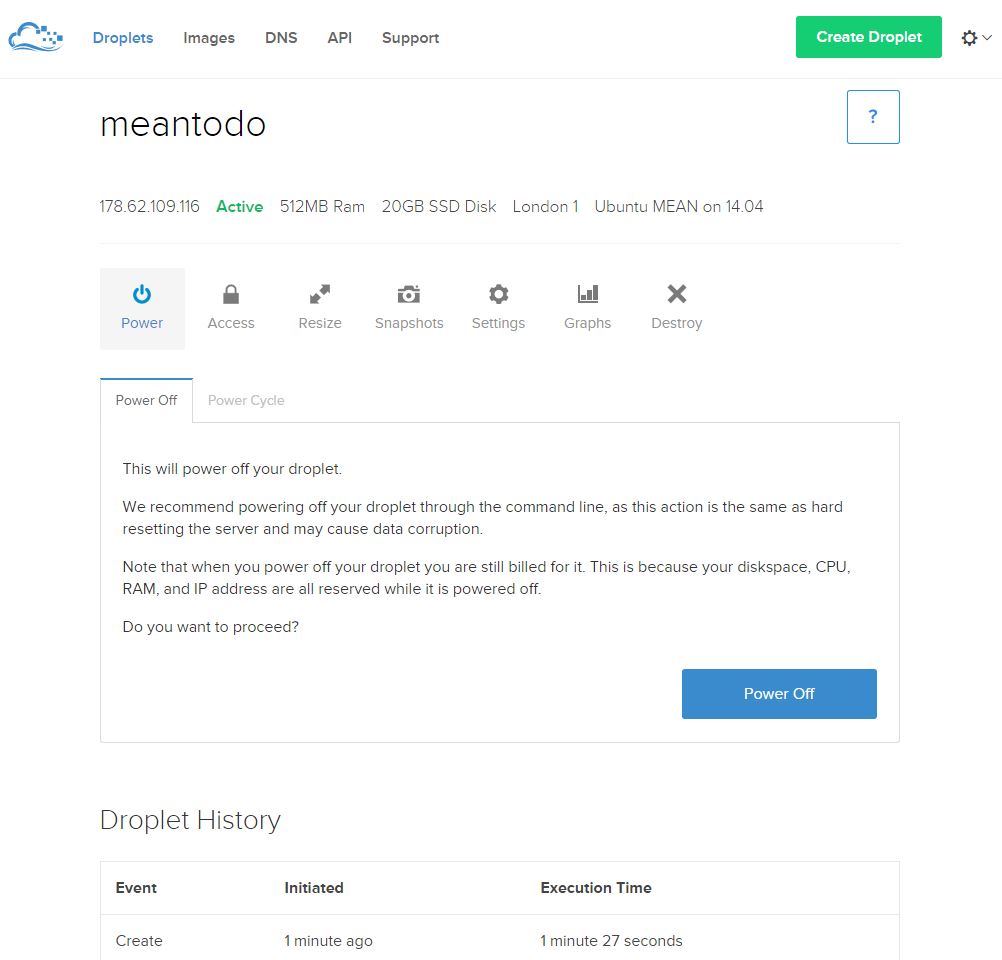
Now comes the most awesome part. In **Select Image** section you have tons of options ranging from from the barebones distributions (Centos, Ubuntu, Debian, etc.), to the pre-built stacks (LAMP, LEMP, Wordpress, ROR, MEAN, etc.) like shown on the image below. Here you have to choose the MEAN on 14.04 option ([YMMV](http://www.urbandictionary.com/define.php?term=ymmv) on the exact number), and this is awesome because with this droplet you won't have to go through the hassle of installing Node.js, Express and MongoDB as we did in the [first tutorial](https://hackhands.com/how-to-get-started-on-the-mean-stack/) - it will be already done and waiting for you! [B-E-A-utiful](https://www.youtube.com/watch?v=UPphDSc6ZEE)!



Then, just wait a few seconds and your droplet will be ready:

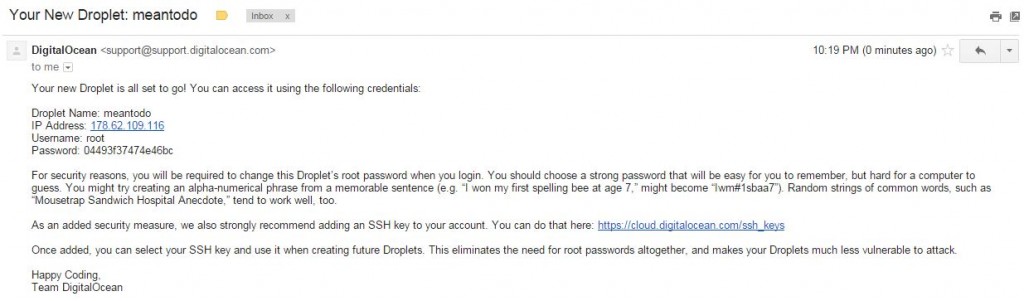


You'll be redirected to the dashboard (once the droplet is created) which will look something like the image below:

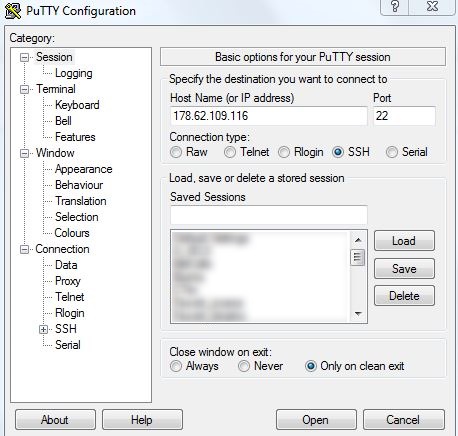


#### Connect to the droplet

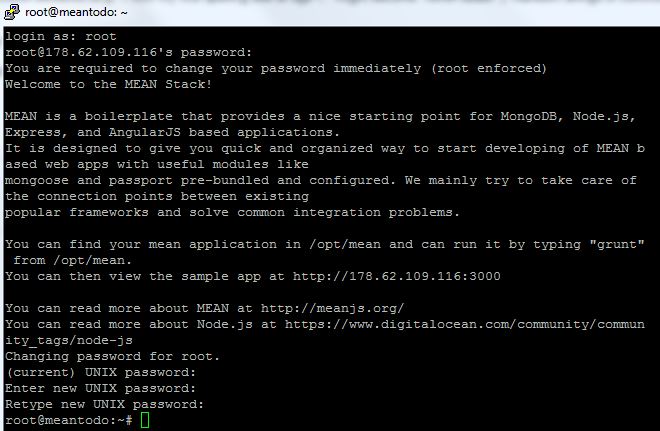
On your email address you will receive instructions which will look something like on the image below:



Now, using a SSH client of your choice (I'm currently on Windows so not much leeway here :)) connect to the droplet. The image below shows the setting from the [Putty](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) SSH client on Windows:



You have to connect with the **root** user and the password that was emailed to you. However, immediately upon logging in you will get a notification that you have to change the root's password, as shown on the image below:



#### Create a new user

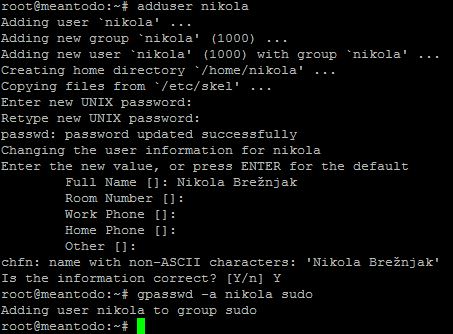
Using root user in linux is bad practice, so now you're going to create a new user with the following command:

useradd nikola

Of course, you can use any username you want. After this add the user to the **sudo** group by executing the following command:

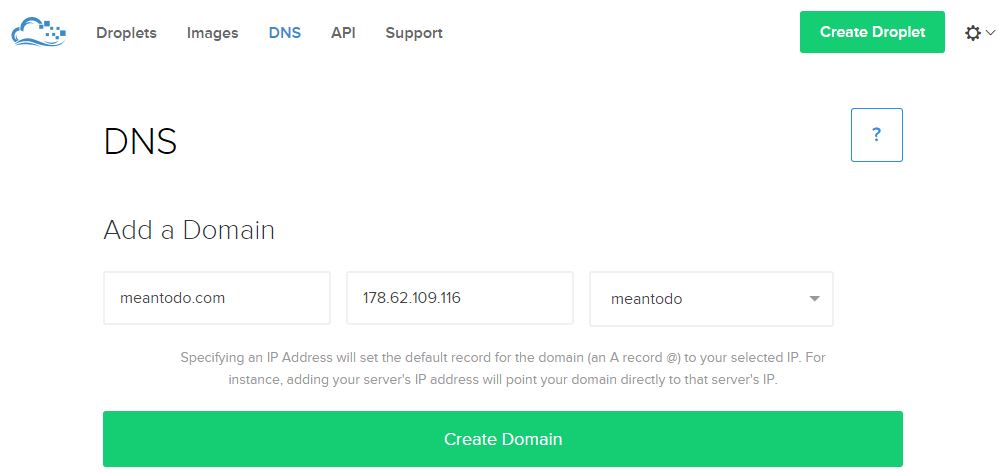
gpasswd -a nikola sudo

The sudo group enables the user nikola to use the sudo command when using some commands that are restricted to root user. You can learn more about sudo [here](https://www.digitalocean.com/community/tutorials/how-to-add-delete-and-grant-sudo-privileges-to-users-on-a-debian-vps). The image below shows the example of the commands I just listed:



#### Use your own domain

In case you bought (or have somewhere hanging in the closet from the [90's .com boom](http://en.wikipedia.org/wiki/Dot-com_bubble)) a domain that you would like to use to point to your droplet you have to go to the **DNS** settings and add a domain like shown on the image below:



Also, in your [domain registrar](http://en.wikipedia.org/wiki/Domain_name_registrar) (where you bought your domain - think Godaddy, Namecheap, Hostgator, etc.) you have to set the corresponding nameservers to:

* ns1.digitalocean.com
* ns2.digitalocean.com
* ns3.digitalocean.com

You can learn more about this on the [official Digital Ocean guide](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-host-name-with-digitalocean).

### Using PM2

Running your Node.js application by hand is, well, not the way we roll. Imagine restarting the app every time something happens, or god forbid application crashes in the middle of the night and you find about it only in the morning - ah the horror. [PM2](https://github.com/Unitech/pm2) solves this by:

* allowing you to keep applications alive forever
* reloading applications without downtime
* facilitating common system admin tasks

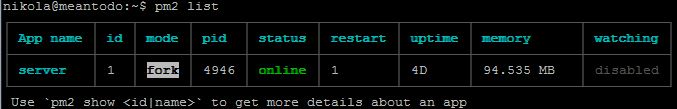
To install PM2, run the following command:

sudo npm install pm2 -g

To start your process with PM2, run the following command (once in the root of your application):

pm2 start server.js

As you can see from the output shown on the image below, PM2 automatically assigns an App name (based on the filename, without the .js extension) and a PM2 id. PM2 also maintains other information, such as the PID of the process, its current status, and memory usage.



As I mentioned before, the application running under PM2 will be restarted automatically if the application crashes or is killed, but an additional step needs to be taken to get the application to launch on system startup (boot or reboot). The command to do that is the following:

pm2 startup ubuntu

The output of this command will instruct you to execute an additional command which will enable the actual startup on boot or reboot. In my case the note for the additional command was:

sudo env PATH=$PATH:/usr/local/bin pm2 startup ubuntu -u nikola

If you want to learn more about the additional PM2 options you can take a look at [this post](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-14-04#other-pm2-usage-(optional)?refcode=974c9bc93d77).

### ****Using NGINX as a Reverse proxy in front of your Node.js application****

Though this step is not mandatory, there are several benefits of doing so, as answered in [this Stack Overflow question](http://stackoverflow.com/questions/16770673/using-node-js-only-vs-using-node-js-with-apache-nginx):

* Not having to worry about privileges/setuid for the Node.js process. Only root can bind to port 80 typically. If you let nginx/Apache worry about starting as root, binding to port 80, and then relinquishing its root privileges, it means your Node app doesn't have to worry about it.
* Serving static files like images, CSS, js, and HTML. Node may be less efficient compared to using a proper static file web server (Node may also be faster in select scenarios, but this is unlikely to be the norm). On top of files serving more efficiently, you won't have to worry about handling eTags or cache control headers the way you would if you were servings things out of Node. Some frameworks may handle this for you, but you would want to be sure. Regardless, still probably slower.
* More easily display meaningful error pages or fall back onto a static site if your node service crashes. Otherwise users may just get a timed out connection.
* Running another web server in front of Node may help to mitigate security flaws and DoS attacks against Node. For a real-world example, [CVE-2013-4450](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-4450) is [prevented by running something like Nginx in front of Node](http://blog.nodejs.org/2013/10/22/cve-2013-4450-http-server-pipeline-flood-dos/).

So, with being convinced that having NGINX in front of Node.js application is a good thing, following are the steps on how to install and configure it. First, update the apt-get package lists with the following command:

sudo apt-get update

Then install NGINX using apt-get:

sudo apt-get install nginx

Now open the default server block configuration file for editing:

sudo vi /etc/nginx/sites-available/default

and add this to it:

server {

listen 80;

server\_name meantodo.com;

location / {

proxy\_pass http://127.0.0.1:1337;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

This configures the **web** server to respond to requests at its root. Assuming our server is available at[http://meantodo.com](http://meantodo.com/), accessing it via a web browser would send the request to the application server's private IP address on port 1337, which would be received and replied to by the Node.js application. Once you're done with the setting you have to run the following command which will restart NGINX:

sudo service nginx restart

You can learn more about additional NGINX setting from quite a [load of tutorials](https://www.digitalocean.com/community/tags/nginx?type=tutorials&refcode=974c9bc93d77).

### Faking RAM with swap

Since we chose the smallest droplet version of 512 MB we can use [**swap**](https://www.linux.com/news/software/applications/8208-all-about-linux-swap-space) to make this droplet perform better. Swap is an area on a hard drive that has been designated as a place where the operating system can temporarily store data that it can no longer hold in RAM. For a more indepth tutorial on this you may want to check out [official DigitalOcean guide](https://www.digitalocean.com/community/tutorials/how-to-add-swap-on-ubuntu-14-04/?refcode=974c9bc93d77). In order to allocate a swap file space execute the following command:

sudo fallocate -l 1G /swapfile

Then restrict it to only root user by executing:

sudo chown 600 /swapfile

With the next two commands you actually setup the swap space and enable it:

sudo mkswap /swapfile

sudo swapon /swapfile

To view the swap space information you can execute:

sudo swapon -s

To enable swap automatically when the server restarts you have to add the following content to the**/etc/fstab** file:

/swapfile none swap sw 0 0

Additionally, there are two more settings worth changing:

* swappiness - parameter which determines how often your system swaps data out of RAM to the swap space. The value is between 0 and 100 and represents a percentage (60 is the default, but we'll use 10 since we're on a[*VPS*](http://en.wikipedia.org/wiki/Virtual_private_server))
* vfscache\_pressure - \_parameter which determines how much the system will choose to cache inode and dentry information over other data (default value is 100, but we'll use 50)

In order to make this change append the **/etc/sysctl.conf** file with the following content:

vm.swappiness=10

vm.vfs\_cache\_pressure = 50

### Other security concerns

If you want to learn more on how to secure your droplet I advise you to go over the steps in the initial server setup with Ubuntu on [official DigitalOcean tutorial](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-14-04/?refcode=974c9bc93d77).

### Run baby run

The fastest way, if you've followed so far, to get this application running on your newly created and configured server is to copy the code to your server from your local machine using, for example, WinSCP, or you could simply clone my finished project from Github:

git clone https://github.com/Hitman666/MEAN\_MVC\_4thTutorial

Then, cd into the directory and just run:

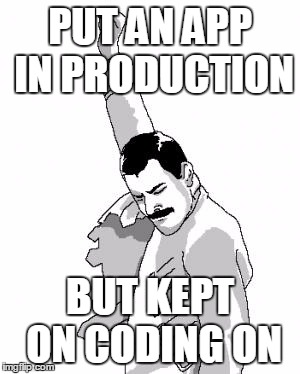
pm2 start server.js

A note though, this will start the so called "development" app. In order to run the "production" app, you have to set the NODE\_ENV variable like this:

export NODE\_ENV=production

and then start the server again. Also, if you'll be using the app in production with your own domain you'll have to edit your Facebook and Twitter application settings to point to that domain, just as we did in the[previous tutorial](https://hackhands.com/mongodb-crud-mvc-way-with-passport-authentication/) (in case you want to refresh your memory how that's done).

## Show must go on



This is it dear people, we've made it. In this 4 part series we've covered a lot of ground, and I really hope you've learned something and that it will be of use to you in your future projects. As I mentioned before, we can take this topic further (testing :O, taking a closer look at [MEAN.JS](http://meanjs.org/) with [Yeoman](http://yeoman.io/) generators are few that come to mind) and if you're interested please share it in the comments below.

Thanks for bearing with me and my MEAN MEMEs. If you feel nostalgic, I made an [image with all of them](http://www.nikola-breznjak.com/blog/wp-content/uploads/2015/06/allMemes.jpg).

If by some chance you would like to download a PDF version of these 4 rather big posts you can do so via[this link](https://leanpub.com/meantodo) (yes, you can choose zero as the amount and I promise I won't take it against you smileyGlasses  
).

hand_rock_n_roll on good people!