The Code Barbarian

The software development blog of Valeri Karpov

Introduction to the MEAN Stack, Part Two: Building and Testing a To-do List

Posted on July 29, 2013 by vkarpov15
In last week's blog post, I showed you how to install all of the basic tools that you need to get up and running with the MEAN Stack. Didn't catch that one and need help getting started with the MEAN Stack? You can find everything you need in Introduction to the MEAN Stack, Part One (http://thecodebarbarian.wordpress.com/2013/07/22/introduction-to-the-mean-stack-part-one-setting-up-your-tools/).

This time we're going to take that shiny new MEAN Stack and put it to good use by building a very simple to-do list. We'll also set up a basic automated testing suite. This process will take 12 steps and you can follow along here (https://github.com/vkarpov15/mean-stack-todo/commits/master). I will link to the diff on github for each step below as well. The *mean-stack-todo* repo started off as a direct duplicate (https://help.github.com/articles/duplicating-a-repository) of the *mean-stack-skeleton* repo from last week.

Over the course of this tutorial, you'll see some of the key features and concepts that make the MEAN stack such a powerful development tool. Here are a few highlights to keep in mind:

- **The MEAN stack comes with a very powerful suite of testing tools.** Javascript's flexibility makes writing tests extremely easy, Jasmine (http://pivotal.github.io/jasmine/) syntax makes your tests into a set of stories (http://misko.hevery.com/2009/09/02/it-is-not-about-writing-tests-its-about-writing-stories/) that help lower the barrier to understanding your code base, and tools like Karma and Nodeunit enable you to have unit test coverage of your entire stack. Even if you decide to not write unit tests for your application, AngularJS's end-to-end test runner enables you to fully automate your integration testing.
- **You can include dynamic Javascript directly into your templates.** The first rule of writing AngularJS code is that your Javascript should never reference the DOM. In the AngularJS world, *$('input#inputName').val('vkarpov')* is (almost) always the wrong thing to do. Your Jade templates should be the definitive guide to all of your app's UI/UX decisions, while your

client Javascript should simply maintain the state of your data.
- **You use the same language all the way through.** MongoDB stores your objects in a format that is very similar to JSON and the database shell uses Javascript-like syntax. NodeJS queries MongoDB using JSON queries, then passes the resulting objects to AngularJS as JSON objects. What does all this mean? All Javascript, all day. This streamlines the code itself and makes coordinating development across your team way easier.

1) Create some sample data for the server to display (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/9baed5a3546e13dd5d34e9ea6f4b298c644a923d)
Our to-dos are going to have a description, a due date, and a boolean flag to indicate whether or not they're already done. Our app.js file shows that when the user accesses the *GET* / route, ExpressJS will call *routes/index.js* "index" function, which in turn renders *views/index.jade*. Let's add a couple of sample to-dos to index.jade's template parameters.

2) Modify the index.jade template to display our sample data using ExpressJS templates (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/4ca0e7cdac304e4da0b84d4fe2452be0f8742a05)
Let's create a very simple layout to display our to-dos. We'll have to separate our to-dos into two lists: one for completed to-dos, and one for outstanding to-dos. It's very important here to note that by default, ExpressJS uses Jade for templating. Jade is a controversial templating library which uses a simplified and more human-readable version of HTML syntax. For example, let's say we have the following HTML:

```
</pre>
<ul class="span6 text-center" id="tools-list">
    <li><a href="http://www.angularjs.org (http://www.angularjs.org)">Angu
    <li><a href="http://www.expressjs.com (http://www.expressjs.com)">Expr
</ul>
<pre>
```

In Jade, this looks like:

```
ul#tools-list.span6.text-center
  li
    a(href='http://www.angularjs.org (http://www.angularjs.org)')
      | AngularJS
  li
    a(href='http://www.expressjs.com (http://www.expressjs.com)')
      | ExpressJS
```

3) Add Bootstrap to our project using Bower (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/0ded5ae648d3de6c7ab30777d86e01879da840ed#commitcomment-3730103)
You'll want to scroll all the way to the bottom of the diff, because this diff includes all of Bootstrap and jQuery. In addition to changes in the layout, you'll notice that we take advantage of some of Jade's inheritance features. The *index.jade* template inherits from *layout.jade* using the "extends layout" on the first line. In this commit, we added "block head" to *layout.jade*, and then in *index.jade* we overwrote "block head" to include bootstrap in our template.

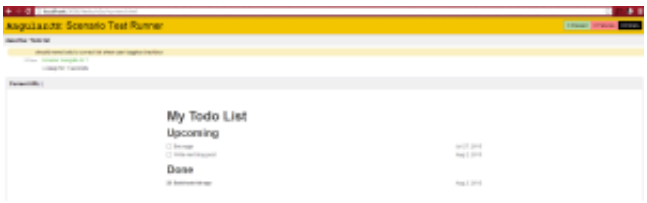
As an aside, you'll want to install bootstrap via

```
bower install bootstrap-css
bower-install bootstrap-javascript
```

because "bower install bootstrap" only gives you an un-compiled version of Bootstrap. This is pretty silly considering the fact that Twitter maintains both Bower and Bootstrap. Fate, it seems, is not without a sense of irony (http://www.quickmeme.com/meme/3vbz9u/).

4) Use AngularJS to make our template dynamic (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/bc89f2c811fd94ef0259a782bcb8cd2c8fd932a7)
One big problem with our to-do list right now is that there is no way to move a to-do from the outstanding list to the completed list, or vice versa. A Jade template is compiled once and then served up as static HTML to the user. However, if we use AngularJS's two-way data-binding as a templating tool, our template updates automatically to reflect the state on the client side. Once you've implemented this, items will automatically move to the completed list when you click the checkbox on the outstanding list.

5) Install AngularJS's end-to-end test runner and write a simple test (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/0dc48bfba23eb62dad51e7546ac4f757c7116cd#commitcomment-3730094)
One of AngularJS's original co-authors was Misko Hevery (http://misko.hevery.com/about/), my former mentor at Google whose sole mission in the professional world is to make sure every piece of software has 100% test coverage. As you might expect, AngularJS was built with ease of automated testing as a core design consideration. Angular-scenario is a part of AngularJS that essentially hijacks an iframe to perform tests on your site as the user would see it. After this commit, run "node app.js" and navigate to http://localhost:3000/tests/e2e/runner.html (http://localhost:3000/tests/e2e/runner.html). You should see something like this:



(http://thecodebarbarian.files.wordpress.com/2013/07/screenshot-linux.png)

This page will run the test we defined in *public/tests/e2e/scenarios.js*. As you can see, Jasmine syntax does a pretty good job of making tests read like stories (http://misko.hevery.com/2009/09/02/it-is-not-about-writing-tests-its-about-writing-stories/). Our test says when we navigate to "/," we should see two to-dos in the outstanding list and one in the completed list. Then we click on a checkbox and expect that one of the to-dos moved to the completed list. You can install angular-scenario via bower

```
bower install angular-scenario
```

6) Write code for a REST-ful path to let the user add a to-do and a unit test (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/c1511cbd6e1d6a3f37f7eaeb7166258642e9348c)

For this step, you need to install nodeunit with

```
npm install -g nodeunit
```

You can run the test suite with

```
nodeunit nodeunit/*
```

from the repo root directory. You'll notice that the route is defined as a function that returns a function. The purpose of this is two-fold. First, it lets you write a proper unit test for this function. The core tenant of writing unit tests is that if you introduce a bug to a clean code base, the only tests that should fail are the ones that test the newly broken function. If the to-do objects were not passed to the route as a parameter, the route would have a hidden dependency that we wouldn't be able to mock out, so bugs in the to-do object itself could potentially break the route's test as well. This would be especially bad if to-dos were a complex object rather than a simple list. The secondary purpose for defining routes in the manner described above is that this syntax allows you to easily see which functions depend on a given module, which is very important if you have to change how a module works.

7) Open up the REST-ful path for adding a to-do (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/67859d39e5f18e2b4762abc39064349c931e9c7e)
In line with the argument in step 6, we'll move the *todos* array into *app.js* for now and inject it into our *GET* / and *POST* /*todo.json* routes. This pattern is called dependency injection (http://en.wikipedia.org/wiki/Dependency_injection).

8) Create a basic form to add a to-do (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/5f108282aa99d750322cf4a2929de3b83454a4d8)
AngularJS gives us the ability to override the default form-submission action using the *ng-submit* (http://docs.angularjs.org/api/ng.directive:ngSubmit) directive, so we can perform the *POST* /*todo.json* action without refreshing the page by using AngularJS's *$http* service (http://docs.angularjs.org/api/ng.$http).

9) Add angular-ui-bootstrap and use its datepicker (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/194efcfb00b5dcfbb7949bb75bfcce3da8da5fe4)
Fair warning, this commit is a little difficult to read because it includes all of angular-bootstrap (http://angular-ui.github.io/bootstrap/). You can install it through bower using

```
bower install angular-bootstrap
```

One critical feature that was hardcoded into the previous commit was the due date of the new to-do. It was set to one week from now and there was no way to change it. Thankfully, the angular-bootstrap module includes, among other features, a nice datepicker directive. The hardest part is setting up an AngularJS module so that we can include angular-bootstrap. AngularJS uses modules to organize dependencies, so we need to create a TodoModule in *public/javascripts/TodoModule.js* and explicitly tell AngularJS that this module depends on the *ui.bootstrap* module (in addition to including the actual ui-bootstrap Javascript file). Once this is done, adding a datepicker input with two-way data-binding to a Javascript date becomes a trivial one-liner:

```
1 | datepicker(ng-model="newTodo.due")
```

10) Set up a MongooseJS model (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/0772a8b15898b671fb156f333e3ba4a9cca2e29d)
MongooseJS is a schema and validation tool for NodeJS and MongoDB. You use MongooseJS essentially the same way you would use an ORM (https://en.wikipedia.org/wiki/Object-relational_mapping) in other development environments. Here we define a basic schema which tells MongooseJS the form that objects in the todos collection must take and define a Todo "model" which allows us to perform ORM-like operations on the todos collection. If you want to learn more about MongooseJS, I've written about it in much greater detail here (http://thecodebarbarian.wordpress.com/2013/04/29/easy-web-prototyping-with-mongodb-and-nodejs/) and here (http://thecodebarbarian.wordpress.com/2013/06/06/61/).

11) Use the Todo model for database persistence and then fix our tests (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/c22f4af1c4ef88ac16c15a87998094cb8bcf8d4a)
Now that we have an object that lets us save and query a todos collection in MongoDB, we can make our *GET* / route query for all to-dos and our *POST /todo.json* route add a new to-do to the collection. Note that the function to save a new to-do can fail – if the posted object takes on an incorrect form (e.g. the "description" is empty), we'll return an error object. For more information about handling this error object, you can read my post How to Easily Validate Any Form Ever Using AngularJS (http://thecodebarbarian.wordpress.com/2013/05/12/how-to-easily-validate-any-form-ever-using-angularjs/).

12) Tie up some loose ends: persist checking and un-checking, and reload data periodically (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/e8fc93e12f68a962ee3dae0eb748e7dc4af62853)
If you've been paying close attention, you may have noticed that up until now we missed a very important feature: we never actually update the backend when we mark a to-do as done. You would also be more observant than I was when I was outlining the steps for this post, because I missed that until I got to this step. However, like most of the tasks in this post, it is pretty trivial. We open up a *PUT /todo/:id.json* route that updates an existing to-do, create a function called "update" in *TodoListController* that uses *$http* to perform a *PUT*, and use the *ng-change* directive (http://docs.angularjs.org/api/ng.directive:ngChange) to call "update" every time the checked/unchecked status of a to-do changes.

Finally, we can add one more feature that I find tragically lacking in my personal to-do list app of choice, Todoist (http://www.todoist.com): periodic syncing with the server. I find it really frustrating to check off something as done on my laptop and then have to refresh the page on my Ubuntu box when I get back home. The simplest way to improve this is to add a dependency to AngularJS's *$timeout* (http://docs.angularjs.org/api/ng.$timeout) service and set it to ping the server for a full to-do list every 30 minutes.

[*Corrected on 8/2/13* : Using *$timeout* causes the E2E test runner to hang until all outstanding calls to *$timeout* finish (https://groups.google.com/forum/#!msg/angular/e2CfWnHjvfU/h9xA5lBkKjsJ). This is a bug that the AngularJS team is currently working on. Until that's fixed, I recommend avoiding using

$timeout in favor of Javascript's native *setInterval* (diff) (https://github.com/vkarpov15/mean-stack-todo/commit/27ecf9bba1b668b7080c098a536207df9639f834), or, better yet, your own custom AngularJS service (http://docs.angularjs.org/guide/dev_guide.services.creating_services) that wraps *setInterval.*]

Conclusion
Hey, guess what? You've just successfully built a to-do list app! How badass is that? What's even better is that in some ways this fancy new app you built is in some ways more useful than commercial offerings like Todoist. *[Note: To any Todoist team members who read this- I love many things about your software but hot damn there are some really annoying issues with it that I think could be improved].* Hope you found this little code adventure useful. By the way, I've been toying with the idea of building a more sophisticated to-do list/task management app in the near future that addresses a lot of the issues I've had working with Todoist, Wedoist (https://wedoist.com/), Asana (http://asana.com/), Jira (https://www.atlassian.com/software/jira), Github (http://github.com/), Trello (http://www.trello.com), Streak (http://www.streak.com/), etc. Would you guys be interested in something like this? Something along the lines of using Github to successfully coordinate tasks that aren't actually code-related. What sort of features would you want to see in something like this? What annoys you about existing to-do or team management tools? Feel free to discuss in the comments section.

Have any questions about the code featured in this post? Want to suggest a better approach? Go ahead and leave a comment below, or tweet me at @code_barbarian. I'm always down for both unfettered praise and horribly vicious flame wars. Until then, feel free to rant about which task management apps you like and which make you want to feel like this (http://rlv.zcache.com/angry_programmer_beats_computer_with_baseball_bat_postcard-r7fea06c5c1584542aac19ef6e3a42f22_vgbaq_8byvr_512.jpg) in the comments. You know what's better than a task management app? A guy like William Kelly (@idostartups). Major props to him as always for helping make this post happen.

Bookmark the permalink.

45 thoughts on "Introduction to the MEAN Stack, Part Two: Building and Testing a To-do List"

Michael Allan says:
on August 2, 2013 at 9:09 am
Hi Valeri,
Thanks for these awesome posts
In section 5, you're missing an image half-way through (where we "should see something like this:" Also, I'm getting 0 tests Passed/Failed/Errors when running the end-to-end tests after checking out the repo. The app works as described if I perform each of the steps in runner.html manually, but the runner itself stalls at the navigation to "/". Likely something to do with my

setup (versions perhaps) … I'll let you know if I get to the bottom of it.
Cheers,
Mike

Reply

vkarpov15 says:
on August 2, 2013 at 1:52 pm
Hi Michael,

Thanks for catching the missing image. Also good call on the hanging E2E tests, that's my mistake, there's a bug in the AngularJS E2E test runner that causes it to wait until all calls to $timeout are finished, which means that the E2E tests don't run at all after step 12. My apologies for missing that, this is what I get for not running my own tests . I added a new commit to simply use setInterval instead (https://github.com/vkarpov15/mean-stack-todo/commit/27ecf9bba1b668b7080c098a536207df9639f834). This isn't an ideal solution because you wouldn't be able to test that using Karma, which is a unit test runner for AngularJS, but it's good enough for now.

Glad you're enjoying the posts!

Cheers,
Valeri

Reply

Seb says:
on August 6, 2013 at 5:10 pm
Thanks a lot for the article and the previous. I'll try to properly go through as soon as I have time. I really like the idea of using MEAN stack but I have a long way to go. Your tutorials are a great place to start so thanks!

I actually like TodoIst and don't find much wrong with it I guess coming from Springpad it is pretty cool.

Reply

Matthew Kim (@iamakimmer) says:
on August 10, 2013 at 4:40 pm
Went through the tutorial, it was an excellent intro!

Reply

vkarpov15 says:
on August 10, 2013 at 7:22 pm
Much appreciated, glad you enjoyed it!

Reply

Gene Conroy-Jones (@gwtsushi) says:
on August 12, 2013 at 3:56 am
Great introduction to the MEAN stack.

Reply
Rolo says:
on August 26, 2013 at 11:48 pm
I'm about to finish the tutorial, still I find that the date from the datepicker is not being passed to the POST, and today's date is not being displayed, when the form is being loaded.

When I clone your repo and compare it to mine, the only difference that I find is that the calendar is being drawn differently.
When I inspect the calendar table element, I find that:
class = ng-pristine ng-valid ng-valid-date ng-valid-date-disabled —> NOT WORKING
Vs.
class = well well-large ng-pristine ng-valid —> WORKING

Everything else worked like a charm, but I still have that thorn of not being able to save the date….

Cheers and thanks a lot for this great series of tutorials.

Reply
Sean says:
on October 8, 2013 at 9:40 am
I've got the same problem.

Reply
Chris Ridmann says:
on August 27, 2013 at 8:15 am
Very interesting – I like this approach! Thanks for a great tutorial!

Reply
Katalyst says:
on August 29, 2013 at 3:51 pm
Hi Valeri, wonderful post!

However I have encountered a little problem. When i create a new todo item and set a different date… the datepicker does not pass the value on to the new todo item and all my new todos are defaulted to the +1 day "due" value. Is there something that I might have gotten wrong?

Reply
vkarpov15 says:
on August 29, 2013 at 4:13 pm
Hi Katalyst,

A few people have noted this, I'm gonna take a look at this later today. Do you mind telling me what browser you're using?

Cheers,
Valeri

Reply

rolo says:
on August 29, 2013 at 5:12 pm
I'm experincing the same, using chrome in ubuntu 12.04.

Katalyst says:
on August 29, 2013 at 7:11 pm
I am also running Chrome but on Ubuntu 13.04. I have read elsewhere that perhaps a directive is needed? Not sure about it though

Leigh's Persistent Thoughts says:
on September 3, 2013 at 12:54 am
Hi Valeri,

I'm experiencing the same – I've tried with the same results in IE10, Chrome 29 and FF 23, all on Windows 7.

I've tried Kiri's suggestion below too, fixing quotes, but it seems to give me no datepicker nor anything else below the last textarea.

kiri says:
on September 3, 2013 at 6:09 am
The issue with the datepicker can be solved as following
Insted of:
datepicker(ng-model="newTodo.due")
use:
div(ng-model="newTodo.due"): datepicker()

Reply

nickb says:
on April 21, 2014 at 11:50 pm
Thanks @kiri, that did the trick.

Rolo says:
on August 29, 2013 at 8:07 pm
in my opinion, it seems that there should be something different in the angular-bootstrap package, since the code I have tested is exactly the same as in the final version that is on github, that's whay I think this is odd, since it's working there…

Reply

akmanocha says:
on August 31, 2013 at 5:57 pm
I qam stuck after step 4. I was expecting the index page should give me the items in the todos separated in updoming and done, 2 and 1 item respectively.
But I am not getting anything. Just 3 blank headers To Do list, updcoming and Done. Nothing in between

Reply
akmanocha says:
on September 1, 2013 at 8:54 am
here is the diff I can observe when i run my step by step app and the completed mean-stack-todo
abhishek@abhishek-3000-N100:~/mytestapp2$ node app.js
Express server listening on port 3000
GET / 200 1096ms – 1.49kb
GET /stylesheets/style.css 304 6ms
GET /javascripts/vendor/angular/angular.js 304 4ms
GET /javascripts/controllers/TodoListController.js 304 4ms
GET /javascripts/vendor/bootstrap-css/css/bootstrap.css 304 164ms
^Cabhishek@abhishek-3000-N100:~/mytestapp2$ cd ../mean-stack-todo/
abhishek@abhishek-3000-N100:~/mean-stack-todo$ node app.js
Express server listening on port 3000
GET / 200 1640ms – 1.97kb
GET /javascripts/TodoModule.js 304 3ms
GET /javascripts/vendor/angular-bootstrap/ui-bootstrap-tpls.js 304 17ms
GET /javascripts/controllers/TodoListController.js 200 19ms – 1.21kb
GET /javascripts/vendor/bootstrap-css/css/bootstrap.css 200 28ms – 124.37kb
GET /stylesheets/style.css 200 52ms – 110b
GET /javascripts/vendor/angular/angular.js 200 79ms – 481.87kb
GET /todos.json 200 9ms – 347b
GET /javascripts/vendor/bootstrap-css/img/glyphicons-halflings.png 304 2ms
^Cabhishek@abhishek-3000-N100:~/mean-stack-todo$

As I expected for style.css angular.js TodoListController.js bootstrap.css all are giving me 304
So, I guess I am missing some config

Reply
akmanocha says:
on September 2, 2013 at 7:50 am
Well, excuse my earlier comments. #304 was just an oversight, got confused with 404. But definitly there something I am missing or overlooking with angular – express integration, can you guys please help.

On google I have seen other resources for the help too. all what is required is saying ng-controller and ng-init (though there I can try setting todo = instead of calling settodos , i havent tried this) and then there are ways to specify that in app.js only instead of routes/index.js Would that make a difference? I am not sure about exports too.

I am new to MEAN, so might be I am doing something silly but I guess no question is silly question.

Reply
Giu says:

on September 2, 2013 at 1:22 pm
Very interestings posts, perfect because I want to enter in nodeJs for our projects.

Do you think MEAN is a good environment for business development?(ERP,CRM, etc..)

There are some important difference between this and mean.io? http://www.mean.io/

Reply
kiri says:
on September 2, 2013 at 2:37 pm
The issue with the datepicker can be solved as following
Insted of:
datepicker(ng-model="newTodo.due")
use:

Reply
kiri says:
on September 2, 2013 at 3:21 pm
Insted of:
datepicker(ng-model="newTodo.due")
use:
div(ng-model="newTodo.due"): datepicker()

Reply
adam says:
on September 15, 2013 at 1:56 am
Thanks for the tutorial, it was definitely very helpful in trying to get up to speed. I'm wondering if you know what it would take, or if it's even possible to remove jade from the project entirely? I want to continue to use html5 and learn angular without jade. I want to extend your tutorial app a bit more to learn the internals and externals of mongo/ose, angular and express.

I tried simply re-writing the index.jade and layout.jade into one index.html file and removing some jade references, then doing a node uninstall jade, but it seems to be 'built-in' pretty heavily. Do you know if there's an easy way to get it running without jade?

Reply
Lujaw says:
on September 27, 2013 at 4:45 am
Thanks for an excellent tutorial vkarpov15

Reply
Bastien Chamagne (@doobinay) says:
on October 28, 2013 at 9:02 pm
Hey Valeri,
thank you for that excellent introduction to the MEAN stack.

I have one question about it.
Is there an easy way to share server-side and client-side model validation? That would be awesome, so we don't have to make a request if it's not valid!

Reply

vkarpov15 says:
on October 28, 2013 at 9:51 pm
If you want to do validation on the client side, thats a bit trickier, but if you read http://thecodebarbarian.wordpress.com/2013/05/12/how-to-easily-validate-any-form-ever-using-angularjs/ you'll see that can ask the server to validate for you in a pretty generic way.

Reply

Bastien Chamagne (@doobinay) says:
on October 29, 2013 at 3:53 pm
Thank you Valeri, server-side validation is fine for now. When I'll be in a more advanced part of my project, I'll think of a way. I know Breeze.js do that, but it looks difficult for me right now. (Learning too much, too fast)

theprofessor117 says:
on February 14, 2014 at 1:33 am
Hi Valeri,
Awesome article! Extremely helpful. I have a question regarding the templating. You mention above that your jade templates should be the guide for UI/UX decisions. My question is, what types of templates do we template from the server, and what should be templates on the client? In other words where should we use jade and where should we use HTML in the public folder?

Reply

theprofessor117 says:
on February 14, 2014 at 1:40 am
Hi Valeri,
Awesome article! Extremely helpful. I have a question about the templating. You mentioned above that the jade templates should be your guide for UI/UX. My question is when should I use the server-side templating and when should I be using client side? In other words what should I use jade templates for and what should I use the HTML in the public folder for?

Thanks again for the great post,
Tom M

Reply

vkarpov15 says:
on February 14, 2014 at 3:20 pm
Hi Tom,

Glad you enjoyed the article. The answer to your question is that you should use Angular for templating as much as possible for flexibility. However, HTML in the public folder should generally be for truly static code. Putting that code in a route gives you more fine grained control with logging and keeping HTML in memory, whereas you need some extra configuration to make Express do anything other than just read the HTML directly from disk if you serve up your HTML directly from public/. Hope this helps.

Reply

theprofessor117 says:
on February 15, 2014 at 12:23 pm
Thanks a lot Valeri! That's very helpful.

Guy Gascoigne-Piggford says:
on March 1, 2014 at 8:57 pm
Just stumbled upon your blog and and worked through this example, really nicely done, clear and very easy to follow. Thank you.

Reply

Daniel Lam (@danie11am) says:
on March 23, 2014 at 12:47 pm
Great tutorial! I followed it and learnt a lot. I wrote a blog post about it and also hosted the web app in my blog, hope you're cool with it! http://daniellam.me/blog/getting-started-on-node-js-and-mean-stack/

Reply

Marco says:
on April 6, 2014 at 8:50 pm
Hi Valery, I'm stuck on step 5 – where we should be able to see the test runner. In fact, I see the header "AngularJS Scenario Test-Runner", but I don't see the todo list… Any idea folks ?

Sorry, I'm a beginner with Mean… I figured out to complete all the tutorial and everything is working apart of the test runner Which is the part I was looking for as a QA guy

Reply

vkarpov15 says:
on April 23, 2014 at 9:15 pm
Hey Marco,

You should see something like this: http://i.imgur.com/1fMM8rI.png when tests run, so you shouldn't see the todo list after the test runner is done. ng-scenario as-is doesn't work on <= IE8, so don't use this with old IE versions. Hope this helps.

Reply

Diego M. says:
on April 6, 2014 at 10:13 pm
nice job!!…I hope one day you can do an article how to deploy an application with the "Mean stack". I tried to do with this app using MongoDB and Heroku, but I'm having problems

Finally, thank you very much for the tutorial, it was a great help for me and running all perfectly.

Reply
kyleventures says:
on April 25, 2014 at 9:43 pm
Thanks for the fun tuts.

Reply
Como Ganar a La Ruleta – Tips on How to Win at Roulette says:
on May 14, 2014 at 1:40 am
I every time used to study piece of writing in news papers but now as I am a user of web therefore from now I am using net for content, thanks to web.

Reply
mohanlate says:
on May 23, 2014 at 7:02 pm
Can I add another controller? If so, how?
How to register a new controller?

Reply
vkarpov15 says:
on June 3, 2014 at 4:23 pm
You should be able to simply create another function, e.g.

function OtherController($scope) {
// Stuff
}

And then use `ng-controller="OtherController"` in your HTML

Reply
click here says:
on May 28, 2014 at 9:33 pm
You actually make it appear so easy together with your presentation however I to find this topic to be actually one thing which I feel I'd never understand. It seems too complex and extremely wide for me.

I'm looking forward to your subsequent publish, I'll try to get the cling of it!

Reply
Rafael C says:
on June 19, 2014 at 11:58 pm

@vkarpov15 [With tests, NICE ! hehe]
Thank you !
On my setup (ubuntu-14.04 based – chrome)
I had to move the datepicker out of the form because the arrows to change month were submiting the form.
I forked but I dunno if everyone had this problem, so I didn't make a PR.

Well, thank you again ! ! !

Reply
David says:
on August 19, 2014 at 4:37 pm
Great post! Would you ever consider revamping this tutorial so that it covers the latest version of Node? Some things have changed in terms of syntax and it's difficult to follow along when these versions are incompatible. Thanks!

Reply
vkarpov15 says:
on September 4, 2014 at 9:38 pm
Do you mean Node 0.11.x and ES6 syntax, like yield and such? Probably not in the near future: Node 0.11 is unstable, and ES6 support on browsers is spotty, meaning your code sharing options are limited even if you're using node 0.11.

Reply