Building an Angular and Express App Part 2

elopers Are So Powerful How I Reg

Posted on 16 April

by J Cole Morrison

Note: This is a continuation from Building an Angular and Express App Part 1 so make sure you check it out before you do this one.

Also checkout Building an Angular and Express App Part 3.

Alrighty! Today we're going to do a few of things:

1. Get the angular front end workflow setup

3. Get a local mongodb instance setup

I'm trying to keep these somewhat short, since I know that my own attention span waxes and wanes tremendously when following code tutorials. We won't be touching on database access or security just yet. The reason is that those two topics will DEFINITELY need their own post. I want to explain what's happening in them thoroughly since they will be the LEAST written about (where as there are plenty of Angular JS tutorials).

Project Setup and Cleanup

Before anything, let's checkout a new git branch. For those unfamiliar, we're just making a "duplicate" of our code so that if we royally screw up we can revert back easily. Do the following in your root project folder:

```
$ git checkout -b angular-express-tut-part2
```


Next, we need to clean up our server directory. We just need to remove the Jade templating engine since we won't be using it. Make sure you have your two command line tabs open. One in your project's `client` directory and the other in the `server` directory. Neither your grunt or your express server should be running.

In the `server` tab run the following:

```
$ npm uninstall jade --save
```

This will remove it from our `server/package.json` and also remove the Jade folder and files from our node_modules. Once we deploy, that package.json will be key to making sure we get all of the right stuff.

Some Front End Prep

This first bit is just going to make things a bit more.. appealing, so that we don't have to look at the ugly default bootstrap colors. This is also how I go about starting one of these projects. I tend to define the View before working on the Model or View Model (controller).

0) Open your `client` command line tab and do a `$ grunt serve` and open up your `server` tab and do `$ npm test`. Close the tab that Grunt opens up `127.0.0.1:9000` and navigate to `localhost:3000` in your browser.

1) Open up your `client/app/styles/_variables.scss` and replace lines 20 to 34 with the following:

```
$brand-primary:         #404040 !default;
$brand-success:         #04BF9D !default;
$brand-info:            #34D0CA !default;
$brand-warning:         #f0ad4e !default;
$brand-danger:          #F53D54 !default;
$brand-dark:    #404040 !default;



//== Scaffolding
//
// ## Settings for some of the most global styles.

//** Background color for `<body>`.
$body-bg:               #fbfbff !default;
//** Global text color on `<body>`.
$text-color:            $brand-dark !default;
```

2) Open up your `client/app/views/main.html` view and replace the code on line 7 with the following:

```
<p>
    <a href="#/signup" class="btn btn-lg btn-success">
    Signup
    <i class="fa fa-chevron-circle-right"></i>
    </a>
</p>
```

All we're doing here is changing the text inside of the button AND prepping it as a link to our route we're about to setup. Note the hashtag (#) in our href. This is there to make it backwards compatible with that miserable, terrible abomination known as Internet Explorer. Yes you can get rid of it, BUT in all the production apps I've done, there's just been too many headaches from people telling me that it's "not working." I realize what most stats say very few use IE..... but I don't believe them.... because my experience is overwhelmingly otherwise.

Refresh your page on `localhost:3000` and you should see your new color scheme. Can't really see the new colors yet because we're only leveraging a couple of them.

Setup the Signup Route in Angular

Now we get to see some Yeoman magic. If you haven't used it before or haven't experimented with our last project this is really cool. Open up another command line tab and navigate to your project's `client` folder. If you're on Mac, just open your current `client` tab where the grunt server is running and press `cmd + t`. Run the following command:

```
$ yo angular:route signup
```

And all sorts of magic will begin to happen. Yeoman will create all of the files you need, append the route to your `client/scripts/app.js` and save you about 5 - 20 minutes of your life depending on how bad you are with typos. Refresh your page and click on your signup button to navigate to your new route. Alternatively, just point your browser to `localhost:3000/#/signup`.

All we're going to do right now is create the base form we'll be using for the signup process.

1) Create a new file - `client/app/styles/partials/_signup.scss` and add the following code to it:

```
.signup-container{
    margin-top: 5em;
}

.signup-header{
    text-align: center;
    color: $gray-light;
    margin-bottom: 1em;
}
```

2) Open up `client/app/styles/main.scss` and add the following line right after the `// endbower` comment:

```
@import "partials/signup";
```

3) add the following code to your `client/app/views/signup.html` file. (the gist also includes code from the `_signup.scss` file too)

The Gist

4) Restart your Grunt server (the one in the `client` tab). Sometimes, when adding new directories to our SCSS, Grunt will get hung up. All you need to do is restart it if this happens.

5) Refresh your page and check out your shiney new form.

The Angular Signup Controller

What happens now is that we need to angularize our form. I'll give a quick run down of how this works, but learning the basics of Angular should probably be done elsewhere.

In our form in `signup.html`, each `<input>` element is going to get an `ng-model` attribute. This attribute allows Angular JS to become aware of anything that we type into that input and access it. Our `<button>` element is going to get an `ng-click` attribute that will specify a method that should be run when it's clicked. Use the code from the following gists in your `client/app/views/signup.html` and your `client/app/scripts/controllers/signup.js` files:

The Signup Html and Js

I try and explain what's happening in the code comments. `$scope` serves as the data for our current view's controller. For example, in our HTML file, the `<input>` for our `id="firstname"` has an attribute `ng-model="signup.user.firstname`. This data resides within its controller's `$scope` property. Make sure you read through the code comments in the `signup.js` file.

(sorry that the tabs are so far out... github wouldn't let me change it to something lower??)

To confirm that everything is working, make sure both of your servers are running and navigate to `localhost:3000/#/signup` and signup! Alternatively, just go to `localhost:3000` and click on the signup button and then sign up.

If you open your browsers console, you should see the data you filled in get logged.

Getting the Data to Express

Now we're going to send that data over to our express server. To do so we need to setup our signup route. The way we do this will also outline how we'll setup new routes from now on.

1) Create a new file at `server/routes/signup.js`. Don't put anything in it just yet. It won't make sense.

2) Open up your `server/app.js` file.

3) Right before the `var app = express();` line, input the following:

```
/**
 * Route Imports
 */
var signup = require('./routes/signup');
```

4) At the bottom of your `server/app.js` file, just before the `module.exports = app;` line, input the following:

```
/**
 * Routes
 */
app.use('/signup', signup);
```

For those who are unfamiliar with express, the way that it works, is that when an request occurs, it starts at the top where `var app = express();` is and pipes through every method you see listed. The request may or may

not use something from each of the methods. When the method is complete, it passes it on to the next one until we send something to the user.

So when our POST request from Angular hits our Express server, it will pipe through until it gets to our `app.use('/signup', signup)` call. This will then hand the request off to our `server/routes/signup.js` module we're about to fill out.

5) Open up your `server/routes/signup.js` file and add the following:

```
// Include Express
var express = require('express');
// Initialize the Router
var router = express.Router();

// Setup the Route
router.post('/', function (req, res) {

    // show the request body in the command line
    console.log(req.body);

    // return a json response to angular
    res.json({
        'msg': 'success!'
    });
});

// Expose the module
module.exports = router;
```

Something a little weird here..... why is the `router.post` going to `'/'`??? This part of Express 4.x's new routing system. What happend is that in your `server/app.js` file, you told this module to be used when the route `/signup` occurred. SO, in this file, it treats `/signup` as `/`. For example, if you wanted to have a `/signup/special` route you wouldn't touch anything in your `server/app.js` file. You'd just add the following in the `server/routes/signup.js` file:

```
// Setup the route for handling 'special' posts
router.post('/special', function (req, res) {
    res.json({
        'msg': 'this was posted to /signup/special'
    });
});
```

Make sense? Still don't think I like it as much as Express 3.x's router though ...

6) Open up your `client/app/scripts/controllers/signup.js` file and change the `.success()` and `.error()` to the following:

```
request.success(function (data) {
    // our json response is recognized as
    // the data parameter here. See? Our msg
    // value is right there!
    console.log(data.msg);
});

request.error(function (data) {
    console.log(data.msg);
});
```

Here's the Gists for all of these files:

the server app.js and signup route and the clien signup.js

Open up your signup route and use it. You'll see your console logs in the browser show your user info and a message saying 'success!' and you're server command line tab will show the data you just posted to.

Do Some Git

Turn off your 2 servers and do the following in your root project directory:

```
$ git add -A
$ git commit -m "Connected angular to express"
```

Now you're saved your progress. If you want to go back to the beginning just do:

```
$ git checkout master
```

And you'll see all of your code revert back to before we began. To come back to us, do

```
$ git checkout angular-express-tut-part2
```

And your code will update to where we are. woot. We'll mess around with merging features into the master later. This is good for now though.

Setup a Local MongoDB

Alrighty, this is a short one...

...assuming you've got MongoDB installed on your computer. Reference this to get setup:

Installing MongoDB

The point of this part is to prep for the next article. But it's also a nice little workflow technique. I doubt you want one local instance of MongoDB running for EVERY app you develop on it. Thus, intead we're going to create one JUST for this project.

You can close out all but one command line tab. Navigate to your root project folder and do the following:

1)

```
$ mkdir -p data/db && mkdir data/logs
$ touch .gitignore && echo "data/" >> .gitignore
```

The first just makes the directories that will be housing your mongo data. The second makes it so that Git will ignore our database. We don't want it reverting all the time when switching between features.

2) Confirm that it's working

From now on, you'll start up your MongoDB for this project with the following command:

```
$ mongod --dbpath data/db/ --logpath data/logs/mongodb.log --
logappend
```

(for those on a small screen, that's one line)

This tells mongo to use the database in our `data/db` directory instead of the default one that sets up. It also tells it to output logs to our local

directory vs teh default one. The `--logappend` method means that mongo will keep the same log file and not overwrite it.

When we begin connecting node up to our MongoDB instance, we'll always have to run the above command first. Otherwise it'll just freak out and say that it can't connect.

When the above command is used (aka when MongoDB is running) use the following command to access the database in the mongo shell:

```
$ mongo
```

Yep, that's it. No extra options or flags. It'll realize that you want it to access that one.

That's all for now folks!

About the author

J Cole Morrison

Sacramento, CA

http://jcolemorrison.com

Comments

19 Comments start.jcolemorrison.com 1 Login

Recommend Share Sort by Best

Join the discussion…

rambutan • 10 months ago

fantastic work here!

Reply • Share ›

lexan • 2 years ago

3) add the following code to your client/app/views/signup.html file.

I do not understantd. Maybe same thing miss. Maybe HTML. ???

Reply • Share ›

Gorlan Yuen • 2 years ago

Your link at the top that is supposed to link to part 3 is actually linked to part 1. Made me think there was no part 3 for a minute :P

Reply • Share ›

Cole Morrison Mod ➚ Gorlan Yuen • 2 years ago

Ahh, thanks for the heads up! I changed it.

Reply • Share ›

Piyush Rahate • 2 years ago

This is really going good. Just one question,

When we run Mongodb, there would be two tabs for Mongo, one in which we will kick-up the server itself using the mongod --dbpath command, and second where we would run the mongo command. Am I right?

Reply • Share ›

Cole Morrison Mod ➚ Piyush Rahate • 2 years ago

Yep, but you can also run the "mongod" process in the background detached from terminal if you'd like:

http://docs.mongodb.org/manual...

Reply • Share ›

Jameson Trinker • 2 years ago

This might be a really stupid question, and I probably missed a step, but in step 2 when you say to replace the code on line 16 of views/main.html in the client directory with the paragraph tag including the signup button...line 16 is just a header that says "angular". main.html is just the yeoman welcome page. Did I miss something where we replaced main.html with another file, because the context doesn't make any sense? Thanks!

Reply • Share ›

Cole Morrison Mod ➚ Jameson Trinker • 2 years ago

Sorry for the delayed response! I haven't been able to update this one yet for the newest version of generator-angular. In 0.9.x they moved the files structure around a good deal. I should be able to update it here soon.

In the new version, they moved the navigation and footer out into index.html and made the client/app/views a lot smaller. So, as Piyush commented, yeah you just want to replace the button.

Reply • Share ›

Jameson Trinker ➚ Cole Morrison • 2 years ago

Thanks! Mind commenting when you update this? I appreciate it.

Reply • Share ›

Piyush Rahate ➚ Jameson Trinker • 2 years ago

I think you are referring "client/app/views/main.html" and the line number is 7 not 16. You should see a

tag with success button which needs to be update.

Reply • Share ›

Luci3n • 2 years ago

Got to admit this series of tutorials has been great!! :)

Reply • Share ›

Clark • 2 years ago

Question why did you choose to validate the forms like you did with the massive if block vs a method like this http://scotch.io/tutorials/jav...

Reply • Share ›

Cole Morrison Mod ➚ Clark • 2 years ago

A few reasons:

1) even though there's less in the JavaScript in that method, there a great deal more markup. Once one has fleshed a bigger page, with custom alert boxes, ng-repeats, shows, classes, switches, ifs etc will be all over the markup and it can get pretty hairy looking.

So it's kind of a massive JavaScript block (which can be cut down if you do a form serialize, if you're not going back to IE8) vs a ton more markup.

2) less stuff for angular to monitor on the scope. In an app I'm working on now, we

generate a spreadsheet that can get up to 500 elements (inputs). Reducing data bindings or places where angular extends an instance with a method can show up there. Although, that shouldn't be a big deal since their extending the prototype, which makes me think it's something like monitoring an additional scope property?

3) I hadn't delved into a serious analysis and usage of the form validation aspects. Because of 1 and 2 I just didn't see the need. Because (1) and (2) got me in the habit of watching where I did stuff, I just didn't do it.

Of course these are just my reasons. #1 is just preference. #2 was just a bottleneck I experienced. #3 was just developed habits.

• Reply • Share ›

Christopher Chisholm • a year ago

I've followed the tutorial from the beginning, but I'm having a problem. In signup.js, signup.submit() is never actually being called. I get nothing on the console and an alert I put in there is never being called. app.js does have the route, the page definitely has the latest code (i can see the alert in the source), and all the names look correct. How would one go about diagnosing the problem? It almost seems like angular just isn't setup properly or something.

• Reply • Share ›

Vishal Shah ➚ Christopher Chisholm • a year ago

I had the same issue until I realized that the sigup.html code in the GIST is missing the controller assignment. Changing the first line from to made it work.

3 • Reply • Share ›

Amit Pandey ➚ Vishal Shah • 6 months ago

Thanks Vishal

• Reply • Share ›

Pascal Heitz ➚ Vishal Shah • a year ago

Wow thank you. I was stuck on this.

• Reply • Share ›

rambutan ➚ Pascal Heitz • 10 months ago

You don't need to assign controller to signup.html --and shouldn't since it was already assigned in app.config. You need to bind the controller scope with 'this'.

Delete:

```
$scope.signup = signup = {};
signup.user = user = {};
```

replace with:

```
signup = this;
user = signup.user = {};
```

then it works!

1 • Reply • Share ›

Richard Gould ➚ rambutan • 9 months ago

That worked for me - thanks!

1 • Reply • Share ›

ALSO ON **START.JCOLEMORRISON.COM**

The Weird Impact of Expertise on

Quick Tip: Organizing Routes in Large