Building an Angular and Express App Part 3

d Impact of Expertise on Ideation Quick Tip: Organizing Routes in Large Express

Posted on 29 April
by J Cole Morrison

This is a part 3 in a series of posts. You can check out the other parts here:

- Part 1
- Part 2

You should go through those to get the most out of this series.

This post is going to cover the following:

- Setting up MongooseJS to act as our object model
- Making the signup page active
- Signing up your users and saving them to Mongo

Note in code examples `...` represents a break in the code where older code is. For example:

```
// Top of our file

... // <-- just represents code in between with no change

module.exports = app;
// Bottom of our file
```

Hopefully that makes sense. If not, PLEASE ask me, I'm more than happy to clarify!

d Impact of Expertise on Ideation Quick Tip: Organizing Routes in Large Express

Get Git Ready

Let's checkout a new Git branch so that we can venture back to this start point if we mess up. Navigate to your root project folder and do the following:

```
$ git checkout -b angular-express-tut-part3
```

For those unfamiliar with Git, pretend that our project is a video game. When we do a `git commit` it's like saving our progress. When we checkout a new branch like we just did, it's like saving our progress in a different save slot. So if we wanted to, we could do a

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

And we'd be right back where we started at the end of part 2. But just in case, we're going to save to a new slot... in this case checkout a new git branch.

Some Project Prep

First, we're going to rearrange our routes. We're going to make them more inline with the "Express" way of turning routes into "mini-applications." In other words, we're just going to get them setup in such a way that prepares our application for scaling better. As it stands, we might wind up with a really long `app.js` file full of routes and route imports. Open up `server/app.js` and do the following:

1) Delete the lines where you imported the routes:

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

2) Delete the one route we made at the bottom of the file just above the `module.exports = app;` portion:

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

3) rename `server/routes` to `server/router`

4) add a new directory `routes` inside of of your `server/router` folder. So you'll have a new directory that reads as `server/router/routes`. This is where we'll be keeping all of our individual routes.

5) move your `server/router/signup.js` into the `server/router/routes/` directory. So it will now be `server/router/routes/signup.js`.

6) move your `server/router/users.js` into the `server/router/routes` directory as well.

7) delete everything in your `server/router/index.js` file and replace it with the following:

```
/**
 * The Index of Routes
 */

module.exports = function (app) {

    // The signup route
    app.use('/signup', require('./routes/signup'));
}
```

8) open up your `server/app.js` file and just above the our `//Error Handling` and `module.exports = app;` lines, require our router:

```
...

/**
 * Routes
 */
var router = require('./router')(app);
```

```
// Error Handling
app.use(function(err, req, res, next) {
    res.status(err.status || 500);
});

module.exports = app;
```

What we're doing here is pulling in our `server/router/index.js` file and passing it the instance of our Express application. We're then adding the signup route to it which will handle anything related to signup. The signup route is now self contained in its own file.

On the command line, make sure you have one tab open to your `client/` folder and one open to your `server/` folder. In the `client/` command line tab, do a `grunt serve` and in the `server/` command line tab, do a `npm test`. Navigate to `localhost:3000`, click on the signup button and fill out the form. Confirm that when you "Signup!" that the `server/` command line tab outputs the information that you filled out in the form.

Checkout what's changed with git:

```
$ git status
```

"Save your game":

```
$ git add -A
$ git commit -m "Reoganized the router"
```

Again, for those unfamiliar with Git, the `git add -A` is saying that we want to add all of our files (and new ones) and changes we've made to be "potentially saved." When we call `git commit` we actually save those

changes. The `-m` is just a short hand that you pass a message about what you did. If you want to see all of the things you've done, type:

```
$ git log
```

And you'll see the progress thus far.

Getting MongoDB and Mongoose Hooked Up

First off, we're using Mongoose because it let's us sprinkle some "relational" style database elements. Working directly with MongoDB isn't "bad" but you wind up with a lot of boilerplate that's just a waste of time. Don't worry, it's well supported not only by a team of great developers BUT by the MongoDB foundation itself. Navigate to your `server/` directory and do the following:

```
$ npm install mongoose --save
```

This will add the MongooseJS node module to your project and save it to your package.json file. While you're at it, in the `server/` directory also do the following:

```
$ npm install bcrypt --save
```

This is what we'll be using to hash and salt our user passwords into the database for security. This a low level module so it may take some time to install/compile. Additionally, I know some people have trouble compiling it for the first time, so if you run into any problems, navigate over the git hub repo and checkout the issues for help. The Bcrypt Repo.

Now we're going to make some new files and directories in the `server/` folder.

- make a new directory `server/database/`
- make a new directory `server/database/schemas/`
- make a new file `server/database/index.js`
- make a new file `server/database/schemas/users.js`

Just like with our router, we're going to use our `server/database/index.js` file as our "table of contents" to interface into our database. Before you do anything else, open up a new command line tab in your root project directory and do the following:

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

This will fire up the mongodb database located in your `projectfolder/data/db` directory and output logs in the `projectfolder/data/logs` directory. If you don't have this running, Node will freak out when we try to connect to mongo. Now that that's done open up your `server/database/index.js` and input the following:

```
/**
 * Our Database Interface
 */
var mongoose = require('mongoose');

// Connections
var developmentDb = 'mongodb://localhost/test';
var productionDb = 'urlToYourProductionMongoDb';
var usedDb;

// If we're in development...
```

```
if (process.env.NODE_ENV === 'development') {
    // set our database to the development one
    usedDb = developmentDb;
    // connect to it via mongoose
    mongoose.connect(usedDb);
}

// If we're in production...
if (process.env.NODE_ENV === 'production') {
    // set our database to the development one
    usedDb = productionDb;
    // connect to it via mongoose
    mongoose.connect(usedDb);
}

// get an instance of our connection to our database
var db = mongoose.connection;

// Logs that the connection has successfully been opened
db.on('error', console.error.bind(console, 'connection error:'));
// Open the connection
db.once('open', function callback () {
  console.log('Databsae Connection Successfully Opened at ' +
usedDb);
});
```

All this file is doing is opening up a connection to our local mongodb. Once you have a production one, you'd just fill in the `productionDb` variable to its URL. When you run `npm start` it will connect to the `productionDb` and when you run `npm test` it will connect to the `developmentDb`.

Now open up `server/database/schemas/users.js` and input the following:

```
/**
 * Our Schema for Users
 */
var mongoose = require('mongoose');
var bcrypt = require('bcrypt');
var Schema = mongoose.Schema;

// Define the User Schema
var userSchema = new Schema({
    firstname: { type: String, required: true },
    lastname: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    profile: {} // for extra information you may / may not want
});

// A method that's called every time a user document is saved..
userSchema.pre('save', function (next) {

    var user = this;

    // If the password hasn't been modified, move along...
    if (!user.isModified('password')) {
        return next();
    }

    // generate salt
    bcrypt.genSalt(10, function(err, salt){

        if (err) {
            return next(err);
        }
```

```
        // create the hash and store it
        bcrypt.hash(user.password, salt, function(err, hash){
            if (err) {
                return next(err);
            }
            user.password = hash;
            next();
        });
    });
});

// Password verification helper
userSchema.methods.comparePassword = function (triedPassword, cb)
{
    bcrypt.compare(triedPassword, this.password, function(err,
isMatch) {
        if(err) return cb(err);
        cb(null, isMatch);
    });
};

// The primary user model
var User = mongoose.model('User', userSchema);

module.exports = User;
```

Finally, open back up `server/database/index.js` and require our new schema at the top and the export the user at the bottom:

```
/**
 * Our Database Interface
```

```
 */
var mongoose = require('mongoose');
var UserModel = require('./schemas/users');


...

// Open the connection
db.once('open', function callback () {
  console.log('Databsae Connection Successfully Opened at ' +
usedDb);
});

exports.users = UserModel;
```

So what we've done here is provide a nice separation of concerns for our database. When we want to interact with our database, we'll just require our database index and work from it in there via exposed `exports`. Now that that's setup, let's move on to actually signing up a user.

Signing up a User and Storing Them

Before we do this, we need to install a couple of extra modules on the server side. They're all for convenience and productivity... but they're terribly convenient and productive! So use them! Go to your `server` command line tab and do the following:

```
$ npm install cli-color --save
$ npm install moment --save
$ npm install underscore --save
```

`cli-color` will let us output colored console logs with absolute ease. `moment` will allow us to output human readable times with ease. Underscore

shouldn't need any introduction.

Note: I use `cli-color` because it doesn't modify any global things. Feel free to manually do coloring yourself. Although, since V8 javascript is so fast you shouldn't be worrying about a couple of support modules bogging you down the least bit.

Open up your `server/router/routes/signup.js` file and put the following:

```
/**
 * This handles the signing up of users
 */
var express = require('express');
var router = express.Router();
var moment = require('moment');
var _ = require('underscore');
var color = require('cli-color');
var db = require('../../database');
var Users = db.users;

// The POST /signup route
router.post('/', function (req, res) {

    // The posted information from the front-end
    var body = req.body;
    // Current time this occurred
    var time = moment().format('MMMM Do YYYY, h:mm:ss a');

    // Check to see if the user already exists
    // using their email address
    Users.findOne({

        'email': body.email
```

```
    }, function (err, user) {

        // If there's an error, log it and return to user
        if (err) {

            // Nice log message on your end, so that you can see
what happened
            console.log('Couldn\'t create new user at ' +
color.red(time) + ' by ' + color.red(body.email) + ' because of: '
+ err);

            // send the error
            res.status(500).json({
                'message': 'Internal server error from signing up
new user. Please contact support@yourproject.com.'
            });
        }

        // If the user doesn't exist, create one
        if (!user) {
            console.log('Creating a new user at ' +
color.green(time) + ' with the email: ' +
color.green(body.email));

            // setup the new user
            var newUser = new Users({
                firstname: body.firstname,
                lastname: body.lastname,
                email: body.email,
                password: body.password1
            });
```

```
            // save the user to the database
            newUser.save(function (err, savedUser, numberAffected)
{

                if (err) {
                    console.log('Problem saving the user ' +
color.yellow(body.email) + ' due to ' + err);
                    res.status(500).json({
                        'message': 'Database error trying to sign
up.  Please contact support@yourproject.com.'
                    });
                }

                // Log success and send the filtered user back
                console.log('Successfully created new user: ' +
color.green(body.email));

                res.status(201).json({
                    'message': 'Successfully created new user',
                    'client': _.omit(savedUser, 'password')
                });

            });
        }

        // If the user already exists...
        if (user) {
            res.status(409).json({
                'message': body.email + ' already exists!'
            });
        }
```

```
    });

});

// export the router for usage in our server/router/index.js
module.exports = router;
```

It's quite the file, but hopefully my code comments help out. Again, feel free to ask. A lot of this file is error checking and logging those errors to your server output. THIS IS IMPORTANT. I know, it's just a tutorial. But seriously, just save yourself some headaches and start doing this now. When you get some user email spamming you about their account not working, it's good to know as much as possible without having to pester them with a ton of questions.

One last quick change in `client/app/scripts/controllers/signup.js`. Change your `.success` and `.error` functions to the following:

```
...

var request = $http.post('/signup', user);

request.success(function (data) {
    console.log(data); // <-- changed
});

request.error(function (data) {
    console.log(data); // <-- changed
})
```

Now! Refresh your page at `localhost:3000/#/signup` and signup! You'll see in your server logs that the new user is being created, and whether or not it was successful.

In your root project folder, run:

```
$ git add -A
$ git commit -m "The end of part 3"
```

And your new branch will be updated. If you want to roll back to the beginning just do a:

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

and of course to come back to the present do a:

```
$ git checkout angular-express-tut-part3
```

If you're ready to turn your master branch into your part 3 branch do the following:

```
$ git checkout master
$ git merge --no-ff angular-express-tut-part3
```

This will make your master branch into your part 3 branch. Now they'll be the same. I'll do this when I know that the feature branch I've been working on is DEFINITE. If you want to checkout a new branch to play around in, just do:

```
$ git checkout -b my-new-feature
```

And you're ready to go.

Recap

This simply setup Mongo up to our Node application. We have it now so that we can sign users up and store their passwords safely. We've also structured our app to be scalable in organization for both routes and database models. Finally, we've continued touching on some workflow with Git and app support issues.

I've put up a git repo of what this looks like on my end when complete. It is missing the `client/app/bower_components`, `client/node_modules` and the `server/node_modules` which can be added by simply running `bower install` and `npm install` in the `client/` folder and `npm install` in the `server/` folder. (This is because git is ignoring those). Here's the repo:

THE REPO

PLEASE feel free to ask me any questions!

Next Time...

We'll be doing security all day.

About the author

J Cole Morrison

Sacramento, CA

http://jcolemorrison.com

Comments

28 Comments start.jcolemorrison.com 1 Login

Recommend Share Sort by Best

Join the discussion…

Amit Pandey • 6 months ago

Great Tutorial, thanks....

• Reply • Share ›

Ranga • 7 months ago

Great series. Request you to please make some time for the Security and Deployment parts also.

• Reply • Share ›

h3ku • 8 months ago

I really like the tutorial, you gonna publish updates? Maybe save session for the user or something.

I want to know how can i control the access to some angular routes depends if the user is logged in or not logged

• Reply • Share ›

James Van Leuven • a year ago

Ok Yoda. Since you're so stellar with all these tutes, do you have any tutes on JWT's? Cuz clearly you are way easy to understand :) (Yes you're a genius)

• Reply • Share ›

Daniel Andersson • a year ago

Seems dead simple to get a get route to work with this setup - but I can only get post routes to work. Is there an example with an working get route under this setup?

▲ ▼ • Reply • Share ›

carr0lls • a year ago

Thanks for the tutorial, that helped a lot but I found a little issue with the underscore omit function used in the signup routes file. You need to set .toJSON() on the savedUser in order to make _.omit() work because Mongoose uses defineProperty. Otherwise, it will fail to omit the key from savedUser.

▲ ▼ • Reply • Share ›

Jordy Meow • Abandoned Japan • 2 years ago

Great series. One thing is missing for me though: deployment. Now that I have everything in my dist, how can I deploy this to a server using git? To Heroku for example? That would be great to know. If that dist was the root, I would know, but I have never "push" a folder separately before. The last final step is missing for me, are you going to write about it? That would be really amazing :) Thanks mate!

▲ ▼ • Reply • Share ›

Mickael Bonfill • 2 years ago

I finally used PassportJS middleware to log in. I have a serious problem with CSRF token during form post from Angular to Express (EBADCSRFTOKEN, status 403). It will be very nice if you could help me to put the good token between the frontend and the backend.

▲ ▼ • Reply • Share ›

Chris Anderson • 2 years ago

Great tutorial, I'd pay to get more.

▲ ▼ • Reply • Share ›

jes • 2 years ago

Nice tutorial!! When can we have the part 4? :)

▲ ▼ • Reply • Share ›

Bart Nice-N • 2 years ago

Hi Cole,
Do you still plan to continue these great series of tutorials? Really looking forward to see the full picture ;)

▲ ▼ • Reply • Share ›

> **Cole Morrison** Mod ➚ Bart Nice-N • 2 years ago
>
> Heya, sorry for the delayed response. While I do definitely plan to continue them, I've just been really busy with other things.
>
> ▲ ▼ • Reply • Share ›

felipe pedroni • 2 years ago

Hey Cole, just a little correction: In the server/app.js is necessary to put the following line in

the "development" enviroment:

app.use(express.static(path.join(__dirname, '../client')));

Without that all the bower_components are not loaded with the "npm test" server. I noticed that part1 of the tutorial had this line, but for some reason it wasn't anymore in your links for app.js at part3·

▲ ▼ • Reply • Share ›

Preston Bernstein • 2 years ago

Hey Cole! Thanks for the walkthrough.

At the beginning of this doc, you ask us to change our routes. I follow the instructions, but it crashes my server. I get this error: https://gist.github.com/ferdyt...

Any ideas?

My code is up on branch angular-express-part3

https://github.com/ferdythebul...

▲ ▼ • Reply • Share ›

> **Cole Morrison** Mod ➚ Preston Bernstein • 2 years ago
>
> Heya! I actually think it's because the new expressjs got rid of the favicon serving middleware (it's not included by default). I'm pretty sure you need to npm install the following and use it in its place:
>
> https://github.com/expressjs/s...
>
> So you would use the above in place of the current favicon serving middleware. Make sense?
>
> ▲ ▼ • Reply • Share ›
>
> > **Preston Bernstein** ➚ Cole Morrison • 2 years ago
> >
> > You're the best!
> >
> > I followed serve-favicon's install directions, but instead changed app.use(favicon(__dirname + '/public/favicon.ico')); to app.use(favicon(__dirname + '/dist/favicon.ico'));
> >
> > ▲ ▼ • Reply • Share ›
> >
> > > **Cole Morrison** Mod ➚ Preston Bernstein • 2 years ago
> > >
> > > ahh, gotcha so it was a location build thing. Sorry about the error, I really need to go through these tutorials and brush them up to the latest versions of things. What's funny is that this tutorial set started because I was doing that a few months ago... and now I get to do them again *shrug*
> > >
> > > ▲ ▼ • Reply • Share ›
>
> **Preston Bernstein** ➚ Cole Morrison • 2 years ago
>
> Wow, thanks for being so quick to reply! Amazing!

I installed that into my node_modules directory, then I edited my app.js file for my server.

Still, no luck.

I've gone ahead and pushed the latest version up to that branch. https://github.com/ferdythebul...

Hopefully you're right, and it's just an easy fix that I'm somehow not seeing.

Reply • Share ›

mdav43 • 2 years ago

firstly thanks for a fine walkthrough. I'm in the process of trying to get this to run on either AWS or google cloud and wondering if anyone has got it to run on either of these platforms?

Reply • Share ›

Cole Morrison Mod ➚ mdav43 • 2 years ago

Hey mdav43, sorry for the REALLY delayed response. I have not tried to get them to run on google cloud yet. On AWS I've used their "Elastic Beanstalk" platform which worked pretty well. However, I've found Modulus.io to be the most straightforward AND cost efficient. They also easily hook you up with Mongo 2.6.3 which comes with full text search! AWS only supports 2.4.9 last time I checked.

Reply • Share ›

Brandon Tate • 2 years ago

Like everyone has already said, thank you very much J Dawg. I've been reading up on all this stuff and trying to figure out the best way to put it all together. Thanks for connecting the dots and giving me somewhere to start with. Bookmarking your site man, so keep em coming :P

Reply • Share ›

Piyush Rahate • 2 years ago

Had been away for couple of days, but this one is so good.
I was able to setup my first MEAN application....yay.
Thanks Cole, for this elaborate series.

One piece of code, I think I am not getting it right or I missed something.
In the "server/app.js" when I remove the following line of code

app.use(express.static(path.join(__dirname, '../client')));

under the development settings section, I loose all the formatting and get a bunch of errors regarding all the javascript files.

Can't figure out what I am missing?

Reply • Share ›

Cole Morrison Mod ➚ Piyush Rahate • 2 years ago

Hey Piyush! In development mode, it enables express to serve the "temporary" files that Yeoman puts together to the client. Ever since Generator Angular updated, its

required now to access the "bower_components" dependencies. There's probably a better work around, I just haven't been able to update this series yet. Sorry for the problems!

• Reply • Share ›

Andrew Blowe • 2 years ago

Hey man great tutorials, I've just started with the mean stack and am following your steps it's been awesome! Can't wait to start adding more onto the app.

• Reply • Share ›

Cole Morrison Mod → Andrew Blowe • 2 years ago

Thanks! Sorry they haven't been updated recently. I'll get back to this one soon, I promise.

• Reply • Share ›

Andrew Blowe → Cole Morrison • 2 years ago

Awesome can't wait! If possible I would like to see some more angular :)

• Reply • Share ›

Mads Konradsen • 2 years ago

Just wanted to say thanks for the series, it's really great with a hands-on approach! Really looking forward to number 4!

• Reply • Share ›

Cole Morrison Mod → Mads Konradsen • 2 years ago

No problem!

2 • Reply • Share ›

ALSO ON **START.JCOLEMORRISON.COM**

The Weird Impact of Expertise on **How to Try Out Angular in your Existing**