

01/31/2019 • 10 minutes to read

## In this article

[Get Your Yeoman on, Yo](#)

[Finding Generators, Yo](#)

[Adding Speakers, Yo](#)

[Layout, Yo](#)

[Wrapping Up](#)

---

October 2016

Volume 31 Number 10

# [The Working Programmer] How To Be MEAN: Exploring Yeoman

By [Ted Neward](#) | October 2016



Welcome back, MEANers.


As I mentioned in a previous column, it's time to pull a comic-book move and engage in a little "retroactive continuity"—a common move whenever the story needs to change its past to better support its future. In this particular case, the ret-con you need to engage in is to make use of some of the tools that you should've used from the beginning, but didn't, because I deemed it necessary to walk through some of the parts in a step-by-step fashion before effectively hiding them behind the tools.

One thing I've heard through some of your e-mails, for example, is how much the MEAN development experience differs from that of the traditional .NET experience. One commenter went so far as to point out that with Visual Studio, you have all the project templates that can take much of the burden of organizing the source away from you. ASP.NET MVC, for example, decided a long time ago what directory controllers would live in, so that any ASP.NET MVC developer walking up to any ASP.NET MVC project will know exactly where everything lives.

On that note, I begin my discussion on Yeoman, the ECMAScript scaffolding tool, which serves the same purpose as the project template facility in Visual Studio. (For those who aren't aware of the etymology of the term "yeoman," one such dictionary definition defines it as "a servant in a royal or noble household, ranking between a sergeant and a groom or a squire and a page." For this reason, the Yeoman tool, documentation and site tend to pass themselves off as a Cockney English bloke.)

## Get Your Yeoman on, Yo

Like almost everything in the JavaScript universe, Yeoman is a command-line tool that's installed via npm. The documentation for Yeoman is available at [yeoman.io](https://yeoman.io), but it's already obvious what the first step will be: "npm install --g yo." Once completed, this will put the "yo" command-line tool on the PATH, and an easy way to check that it's installed is to simply run it. Running "yo --help" brings up the usual list of options, but running "yo" by itself will actually be more interesting: It brings up an interactive command-line menu of options that Yeoman can execute on your behalf, as shown in **Figure 1**.

 Yeoman with Execution Options

**Figure 1** Yeoman with Execution Options

When freshly installed, Yeoman will not have any generators already installed (contrary to my list in **Figure 1** of a few already there), but scrolling up and down (using the arrow keys) will also show a few other options, such as "Install a generator," "Find some help" or "Get me out of here!" Yeoman is nothing if not obvious.

The term "generator" is an apt and appropriate term; like Visual Studio, Yeoman doesn't actually know how to generate anything itself. Instead, it relies on "generators," community-contributed packages that consist of scripts and templates that can interrogate the user for specifics of what to generate. In order for Yeoman to use one of these generators, however, it must be installed on the local system. In addition, in order for it to be installed, you need to figure out which of the almost 4,000 generators (as of this writing) available you want to use.

## Finding Generators, Yo

While it's certainly possible to let Yeoman search npm for generators (one of the options presented by the command line when it's run), it's usually easier to let search engines manage that for you, or to simply browse for the generator on the Yeoman site ([yeoman.io/generators](https://yeoman.io/generators)). Therefore, for example, if you want to make use of Yeoman to scaffold out a new MEAN application, you need to find one that does Angular (v1 for now) and MongoDB. As it turns out, as of this writing, the one that's by far the most

popular, "angular-fullstack," is right near the top of the list. However, if it wasn't, or if there was something different desired (such as a generator that did React, or for a Chrome extension, or even an ASP.NET Core generator), you could use the Yeoman generators page to search for it. For example, you might want to split the code into two projects, one for the back-end Web API, and one for the front-end Angular client. That would suggest that you want an Express-plus-MongoDB project (sometimes called a MEN project), which has a couple of generators available, including "express-api" or "node-express-mongo"), plus another generator that can build an Angular front end.

For now, the generator you want is the "angular-fullstack" generator, because it will generate both client- and server-side scaffolding. Therefore, once you've identified which generator you want, you need to install it using the npm tool. (Yeoman doesn't use yo to manage generator installs; it relies on npm to handle that.) Thus, the next step is to "npm install --g generator-angular-fullstack." Note the "generator-" prefix; this is the convention for all Yeoman generator packages.

Once installed, Yeoman can use it by simply referencing it (without the "generator-" prefix) as a parameter to the "yo" command: "yo angular-fullstack." At this point, Yeoman will pass control of what happens next to the generator itself, and in the case of the angular-fullstack generator, it'll begin to ask questions about what kind of application you want to scaffold out:

- whether to use Babel or TypeScript
- whether to use HTML or Jade (a popular JavaScript HTML templating library) for markup
- which CSS tool to use (raw CSS, Sass, Stylus or Less)
- which Angular router to use (ngRoute or uiRouter)
- whether to include Bootstrap
- whether to include UI-Bootstrap (an extension to Angular for Bootstrap features)
- whether to use Mongoose (which you've seen before) or Sequelize (modeling for RDBMSes) for the models in the application
- whether to scaffold out the Passport authentication code, and if so, for which services (choosing from Google, Twitter or Facebook)
- whether to use Grunt or Gulp (which I'll examine next time) as the project build tool
- finally, which testing tools to use (Jasmine or a collection of several libraries together)

Once the Q&A is done, Yeoman will generate a slew of files (most of which will fall into either the generated "client" or "server" directories it creates). It will then run an "npm install" to pull down all the dependencies for the server and run a "bower install" (which may require installing bower, "npm install -g bower") to do the same for the client. Note that because the various dependencies that are pulled in by these steps will vary

depending on the exact version of the generator (and the various libraries referenced by the generated code), it's possible (likely, even) that these steps will generate warnings.

Once finished, though, you'll have a fully fleshed out scaffolded application. It won't do much, but if you kick it off with "gulp serve" as the generated README indicates (which will require having Gulp installed—"npm install -g gulp-cli"—if it's not already present), and if MongoDB is running on your local box, it'll bring up a page similar to what the ASP.NET MVC project template generates, as shown in **Figure 2**.



The Scaffolded Application Front End

**Figure 2 The Scaffolded Application Front End**

Note that it has full user-management support, including the ability to authenticate using Google, Facebook, or Twitter (though using each of those will require obtaining the appropriate credential tokens or keys from each of those services and specifying them in the configuration directory in the server code) or using e-mail/passwords.

## Adding Speakers, Yo

Because the application you've been building (sort of) has been a speaker-rating system, one of the first things you'll need to model in this new codebase is that of a speaker. And while you could start diving into the generated files to find out where models are declared and so on, it's much easier to let Yeoman help with that: "yo angular-fullstack:endpoint speaker." It'll ask what you want to use for the API endpoint URL, and then go do its thing. This is the command-line equivalent of using Yeoman to do the ASP.NET MVC right-click/Add-Controller thing, and will generate some "emptyish" files in server/api/speaker for you to modify.

It turns out that the angular-fullstack Yeoman generator can do this for a number of different elements of the application, both client-side and server-side. You can see the full list of "sub-generators" by asking the generator itself for a list via "yo angular-fullstack --help".

## Layout, Yo

Because I haven't really explored the client side of things, I'll leave that be for now. But the server side is something I've been exploring for some time, so without further ado, let's go plunging into that.

First, all the server code—not surprisingly—is contained in the "server" directory. This directory contains the main application file, app.js, the list of Express routes in routes.js


and several subdirectories that contain the rest of the server-side code. That includes "api" (where the model- and controller-related code resides), "auth" (which is where the authentication code resides), "components" (which will contain non-API-related components), "config" (configuration) and "views" (which only contains a 404.html file, for when you request unknown URLs).

The auth directory is pretty much done as is; there will rarely be, if ever, a need to wander around in here. As its name implies, Config contains the configuration values used by the rest of the application in much the same way that I built up the "config.js" file in my previous columns. Most of the developer action will happen in the api subdirectory.

Within api, you'll find three subdirectories: the one you just created ("speaker"), one specifically for modeling users ("user") and one that's generated automatically for you by the generator ("thing") as a template or example to follow. Within each of these API subdirectories, you'll find a pattern—a collection of "double-extension" files: thing.controller.js, thing.events.js, thing.model.js and so on. There will also be an index.js file, which serves as a sort of "single entry-point" for the entire subdirectory (it brings together each of the other files together for easy reference from the rest of the directory) and an index.spec.js file, which is used specifically to test the code found in this directory.

So, for example, if you want to indicate that speakers have a first name, last name, and list of topics they like to speak on, you can open the speaker.model.js file, and model it using standard Mongoose facilities (I discussed this in an earlier column; see [msdn.com/magazine/mt683801](https://msdn.com/magazine/mt683801) ), as shown in **Figure 3**.

Figure 3 The Speaker Mongoose Schema

JavaScript	 Copy
<pre>'use strict'; import mongoose from 'mongoose'; var SpeakerSchema = new mongoose.Schema({   firstName: {     type: String,     required: true   },   lastName: {     type: String,     required: true   },   topics: [String],   active: Boolean,   created: {     type: Date,     default: Date.now   }, },</pre>	

```
    updated: Date
  });
  SpeakerSchema
    .pre('save', function(next) {
      this.updated = new Date();
      next();
    });
  export default mongoose.model('Speaker', SpeakerSchema);
```

(Note the use of the “export”; this is a new ECMAScript 2015 feature, as discussed in my September 2015 column at [msdn.com/magazine/mt422588](https://msdn.com/magazine/mt422588) .) However, now that you’ve changed the model for Speaker, you need to update the tests (in `speaker.integration.js`), or else tests will fail when you run them. I’ll leave that as an exercise for you as a way of exploring the code; run the tests with “gulp test:server” to avoid the client-side tests. Of course, you can always explore the API by using curl (“curl localhost:3000/api/speakers,” which will be empty unless you insert a few via POSTing to that endpoint or directly into MongoDB. Note that the generators are undergoing continuous development, so future versions of the generator will set a different default port or URL).

## Wrapping Up

This hasn’t been a particularly code-heavy column, yet you just rebooted the entire application, gained a whole ton of functionality, and essentially brought the application up to the same level (and beyond) from what you’d been building for the past year or so. Gotta love scaffolding! More important, having built all the parts piece-by-piece by hand prior to running the scaffolding, it’s much easier to understand the code as a whole and what’s happening where. For example, opening up `routes.js` will look familiar to the routing table you built by hand earlier, and the `package.json` (in the root of the project directory) will be bigger, though it will basically remain the same as you had been using.

The only new thing, in fact, beyond the use of Yeoman itself, is the introduction of a “build tool” to gather all the pertinent parts together into the right place, and that will be what I discuss next time. Until then, however ... happy coding!

---

**Ted Neward** is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP and has authored and coauthored a dozen books. Reach him at [ted@tedneward.com](mailto:ted@tedneward.com) if you’re interested in having him come work with your team, or read his blog at [blogs.tedneward.com](https://blogs.tedneward.com) .

---

Thanks to the following technical expert for reviewing this article: Shawn Wildermuth

[Discuss this article in the MSDN Magazine forum](#)

