**Kickstart Your AngularJS Development with Yeoman, Grunt and Bower**

This AngularJS tutorial will cover:

* Generating a bare bones AngularJS app with Yeoman
* Using Grunt to speed up development and help perform repetitive tasks
* Using Bower to add third party plugins/frameworks
* Making minor changes to your AngularJS app

Prerequisites

To get the most out of this tutorial we recommend you have the following skills and resources available:

* A terminal and basic knowledge of the command line
* [NodeJS](http://nodejs.org/) and NPM installed
* Fundamental JS, CSS and HTML knowledge

Files

You can find a repo of this tutorial project [here](https://github.com/Haizyfox/sp-yeoman).

Let’s get started yo!

Alright, let’s get this thing underway. The first thing you need to do is install Yeoman, Grunt and Bower. We’re going to use the Node Package Manager to do this all at once. In a terminal, run the following:

npm install -g yo grunt-cli bower

As simply as that, we now have a powerful set of tools at our disposal. I’ll explain each one as we use it.

Yeoman

Yeoman is used to generate the scaffolding of your app for you. It’ll create the basic folders, files and configurations to get you up and running quickly. Not only that but there are some great custom generators available to create apps of a particular kind – we’re going to use the nifty AngularJS generator.

One of the best features of Yeoman is the ability to use custom generators. We’re going to intall the AngularJS generator to help us get up and running with Angular as quick as possible.

Run the following to install the AngularJS generator:

npm install -g generator-angular

Now it’s time to generate a shiny new AngularJS application. In a fresh project directory, run:

yo angular

The generator will ask you a couple of questions. You can answer yes to include Twitter’s bootstrap. Also answer yes to include ngResource. The rest we won’t need for now so answer no.

Sit back (for a few seconds) and watch the generator do its magic. Yeoman will create your files and folders, then it will run bower install (more on this in a moment) and npm install to fetch any dependencies and lastly it’ll perform any mandatory configuration.

What’s in the box?

Let’s take a look at what Yeoman’s given us:

* .bowerrc
* .editorconfig
* .gitattributes
* .gitignore
* .jshintrc
* Gruntfile.js
* **app/**
* component.json
* karma-e2e.conf.js
* karma.conf.js
* **node\_modules/**
* package.json
* **test/**

Let’s go over some of the more important things to notice here:

**app/ directory**  
The app directory contains your static app. It has your html, css and javascript in it and it’s where you’ll spend most of your time developing.  
**package.json**  
The package.json file helps npm to identify our project as well as to manage all of it’s dependencies. It can also contain all sorts of other metadata relevant to your project.  
**node\_modules**  
This one is self explanatory. This is where all the node modules that your project depends on are stored.  
**Gruntfile.js**  
The Gruntfile is a javascript file that is responsible for configuring your project as well as any tasks or plugins that your project requires. For instance, your gruntfile might specify that your project uses Uglify and that you want it to run uglify on a particular directory at build time. More about Grunt in a moment.  
**component.json**  
The component.json file is used to inform the Bower package manager of your projects dependencies as well as other metadata. In recent versions of Bower this file is called bower.json – more on that in a moment.  
**.bowerrc**  
The .bowerrc file is used to pass general config options to bower.  
**Karma files**  
Karma is a testing framework. We’ll use it to run a few tests for our Angular app.

Phew! That’s quite a lot to take in – once you get more familiar with the output of a yeoman generator however, you’ll learn to love the fact that it handles all of this for you!

Let’s add a few more things to our project before we start get on to some actual development.

Bower – A package manager for the web

Before we use Bower, there’s a small bit of config we have to do ourselves. Bower recently changed their naming convention of component.json files to bower.json files so we need to bring our code base in line with that.

The first thing we need to do is make a small change to our Bower config in .bowerrc so open it up and add the following line:

{

"directory": "app/components",

"json": "bower.json" // Add this line

}

What this does, is it tells Bower to use a package’s bower.json file for instructions on how to install that package.

Since we’re using bower for our own project’s dependencies, we’ll need to rename the component.jsonfile in our project root to bower.json as well. A small ask when using such cutting edge technologies :)

**Bower**  
Bower is a package manager. It will help us to quickly find and install our favourite CSS frameworks, javascript libraries and plugins with just a few simple commands.

Ok, let’s give Bower a whirl. Yeoman kindly used bower to install Bootstrap for us earlier, but that was just the Bootstrap CSS. We want all the nifty Javascript widgets as well.

Since we’re building an AngularJS app, we’ll need Bootstrap javascript that works with Angular.

Luckily, the team over at [Angular UI](http://angular-ui.github.io/bootstrap/) have ported all the Bootstrap Javascript into Angular!. Let’s use Bower to install their library.

bower install angular-bootstrap --save

The –save flag tells bower to add this to our bower.json file as a dependency

Fantastic! That was easy wasn’t it? Now, navigate into your app/ directory and let’s see what we’ve got to work with.

Our static app

Take a look at the contents of the app/ directory.

* favicon.ico
* index.html
* robots.txt
* **components/**
* **scripts/**
* **styles/**
* **views/**

**index.html**  
This should be familiar to most of you, this is the core html page of your app.  
**components/ directory**  
The components directory is like the node\_modules directory but for Bower. It’s where all the packages you install with Bower will be kept. AngularUI Bootstrap, for instance, will be in there.  
**scripts/ directory**  
Again, familiar territory to most. This is where your apps javascript is stored. Note that libraries such as AngularJS will exist in the components directory, so scripts/ is for your files that you write!  
**styles/ directory**  
All your css/sass to make your app look moar pretty.  
**Views**  
This nifty folder is where your Angular Templates will reside.

Next up we’ll take a closer look at the AngularJS files

AngularJS

The Yeoman Angular generator has given us the bare essentials: A module, a controller and a view. Let’s take a look at each of those:

**The Module: /app/scripts/app.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | 'use strict';  // Here we set up an angular module. We'll attach controllers and  // other components to this module.  angular.module('testApp', [])    // Angular supports chaining, so here we chain the config function onto    // the module we're configuring.    .config(function ($routeProvider) {        // We use AngularJS dependency injection to fetch the route provider.      // The route provider is used to setup our app's routes.        // The config below simply says when you visit '/' it'll render      // the views/main.html template controlled by the MainCtrl controller.        // The otherwise method specifies what the app should do if it doesn't recognise      // the route entered by a user. In this case, redirect to home.      $routeProvider        .when('/', {          templateUrl: 'views/main.html',          controller: 'MainCtrl'        })        .otherwise({          redirectTo: '/'        });    }); |

**The Controller: /app/scripts/controllers/main.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | 'use strict';      // Here we attach this controller to our testApp module    angular.module('testApp')        // The controller function let's us give our controller a name: MainCtrl      // We'll then pass an anonymous function to serve as the controller itself.      .controller('MainCtrl', function ($scope) {          // Using AngularJS dependency injection, we've injected the $scope variable        // Anything we attach to scope will be available to us in the view.          // In this case, we're attaching a collection of Awesome Things to display        // in app/views/main.html        $scope.awesomeThings = [          'HTML5 Boilerplate',          'AngularJS',          'Karma'        ];      }); |

**The View: app/views/main.html**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | <div class="hero-unit">      <h1>'Allo, 'Allo!</h1>      <p>You now have</p>      <ul>            <!-- Here we use the AngularJS directive: ng-repeat to loop through our awesomeThings          and print them out as list items using the {{}} bindings -->          <li ng-repeat="thing in awesomeThings">{{thing}}</li>      </ul>      <p>installed.</p>      <h3>Enjoy coding! - Yeoman</h3>    </div>  [/js]    <strong>The Index File: app/index.html</strong>  1  <!doctype html>    <html>      <head>...</head>      <!-- The ng-app directive tells angular which module we'll use      for our app. In this case the one defined in scripts/app.js -->      <body ng-app="testApp">        ...        <!-- The ng-view directive specifies that our templates        (such as views/main.html) will be loaded into this div. -->        <div class="container" ng-view></div>          <!-- Here we load AngularJS and the AngularJS resource component -->        <script src="components/angular/angular.js"></script>        <script src="components/angular-resource/angular-resource.js"></script>          <!-- Here we include our own angular scripts -->        <!-- build:js scripts/scripts.js -->        <script src="scripts/app.js"></script>        <script src="scripts/controllers/main.js"></script>        <!-- endbuild -->          ...      </body>    </html> |

Let’s see it in action!

We’re ready to take our first look at our application. Navigate back to the root directory of your app and run:

grunt server

**Grunt**  
Grunt is a powerful, feature rich task runner for Javascript. In brief, it lets you automate repetitive tasks like compiling coffeescript, minifying css, code validation etc. We’ll be using it to do all of that as well as prepare our code for development and deployment.

Grunt is going to whip through our project folder and prepare everything for us such as compiling our included Bootstrap SASS down to css.

After a few seconds a browser window should pop up with your app running and looking all fancy.

Just to be sure, view the source of the page and take a look at the main.css file that’s included. It should be full of Bootstrap code – thanks to the magic of Bower and Grunt.

Let’s change it up

It’s about time to try our hand at making some changes. Since this is Angular, we’ll start with some AngularJS Testing.

Yeoman was kind enough to generate an example test for our controller, so let’s start there.

We’re going to add another thing to our list of awesome things so opentest/spec/controllers/main.js and let’s change our test to expect 4 awesome things instead of 3:

**test/spec/controllers/main.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | 'use strict';    describe('Controller: MainCtrl', function () {      // load the controller's module    beforeEach(module('testApp'));      var MainCtrl,      scope;      // Initialize the controller and a mock scope    beforeEach(inject(function ($controller, $rootScope) {      scope = $rootScope.$new();      MainCtrl = $controller('MainCtrl', {        $scope: scope        });    }));      it('should attach a list of awesomeThings to the scope', function () {      // Change this line      expect(scope.awesomeThings.length).toBe(3);        // To this      expect(scope.awesomeThings.length).toBe(4);    });  }); |

Now we can use another great feature of Grunt:

grunt test

This will run our Karma tests. They should fail because the test expects 4 awesomeThings and we still only have 3. Let’s go fix that to make our tests pass.

Open app/scripts/controllers/main.js and add another awesome thing to the list:

**/app/scripts/controllers/main.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | .controller('MainCtrl', function ($scope) {    $scope.awesomeThings = [      'HTML5 Boilerplate',      'AngularJS',      'Karma',      'SitePoint'    ];  }); |

Save the file and run the tests again:

grunt test

This time they should pass. Now you can fire up the app in your browser (grunt server) and notice that there’s an additional bullet point. Neat huh?

Using our Bower included package

Let’s use the AngularUI Bootstrap library that we included earlier to turn our list of awesomeThings into a dropdown of awesomeThings.

*Important* Since Bower is just a package manager, it’s not responsible for adding our files to ourindex.html file. We need to do that ourselves.

So open up app/index.html and add the following line:

|  |  |
| --- | --- |
| 1 | <script src="components/angular-bootstrap/ui-bootstrap.js"></script> |

Then, as per the Getting Started documentation on AngularUI Bootstrap’s site, we need to add their module as a dependency to our own Angular module

Open app/scripts/app.js and add the ui.bootstrap module as a dependency:

**/app/scripts/app.js**

|  |  |
| --- | --- |
| 1  2  3  4 | 'use strict';    angular.module('testApp', ['ui.bootstrap'])  ... |

Alright it’s ready to use. Now we need to make a few changes to our view:

**The View: app/views/main.html**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <ul>    <li class="dropdown">      <a class="dropdown-toggle">        Click me to see some awesome things!      </a>      <ul class="dropdown-menu">        <li ng-repeat="thing in awesomeThings">          <a>{{thing}}</a>        </li>      </ul>    </li>  </ul> |

We’ve used some bootstrap css classes, and moved our ng-repeat to create menu items instead of just a plain old list.

AngularUI Bootstrap directives work on classes, so simply by adding the dropdown-toggle class to our tag we’ll have a fully functioning dropdown!

We will need to add the Bootstrap UI module to our tests otherwise they’ll fail so make the following changes:

**test/spec/controllers/main.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | 'use strict';    describe('Controller: MainCtrl', function () {      // load the controller's module    beforeEach(module('testApp'));    // load the BootstrapUI module    beforeEach(module('ui.bootstrap')); // Add this line   ...  }); |

**/karma.conf.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | // Karma configuration    // base path, that will be used to resolve files and exclude  basePath = '';    // list of files / patterns to load in the browser  files = [    JASMINE,    JASMINE\_ADAPTER,    'app/components/angular/angular.js',    'app/components/angular-mocks/angular-mocks.js',    'app/components/angular-bootstrap/ui-bootstrap.js', // Add this line    'app/scripts/\*.js',    ... |

Run **grunt test** to make sure everything passes

Now you can open up your app in the browser (grunt server) once more and take a look at your handy work.

Conclusion

Well there you have it! A simple Angular App, a third party JS library, some tests, minification and a whole heap of other goodies with minimal effort!

We’ve only scratched the surface of what’s possible with Yeoman and its fellows – but I hope this will inspire you to whip up a quick AngularJS app the next time you have a great idea!

Look out for more AnuglarJS tutorials and articles on AngularJS best practices coming soon!