Yeoman: Scaffolding Node Applications

In this lab, you will familiarize yourself with Yeoman, a suite of tools that enable quick scaffolding, in-browser dependency management, and mature builds.

# Objectives

You will learn to use the following tools:

* yo, a Node.js scaffolding plug-in framework,
* bower, a dependency management tool for the front-end, and
* grunt, a generic JavaScript task runner that is used to build Node applications.

# Install Tools

The first thing we have to do is to use Node Package Manager, npm, to install yo. Do this by opening a command prompt in the lesson directory and issuing the command npm install –g yo. This will download & globally install the latest version of yo. This will automatically install the tools bower & grunt. Last, install the yo generator for Angular.js with npm install -g generator-angular.

# Lab Steps

Now that the Yeoman tools are installed, we can begin the lab, whose goal is to scaffold an Angular web application, then add our own controller, view & route.

## Create the web application directory

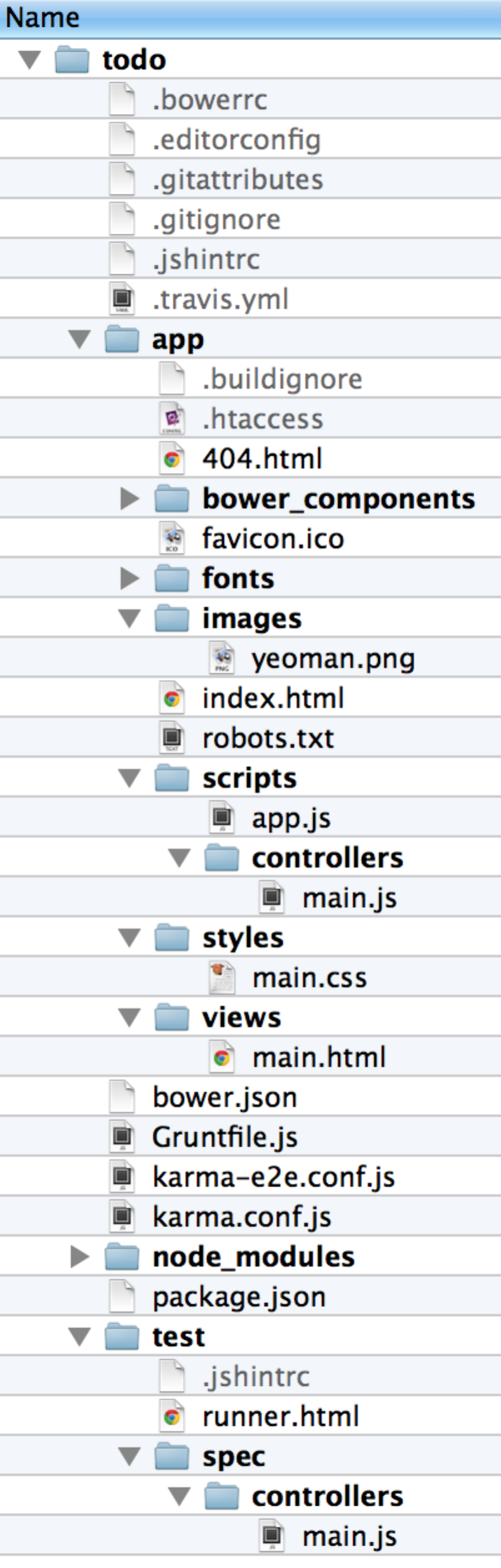
1. Change to this lesson's lab directory and create a directory in it called todo with mkdir todo.

## Scaffold the initial web application

2. Change to the todo directory you just created and scaffold the initial Angular application with the command yo angular.

Choose **not** to use Compass, and then take the defaults for the remaining prompts.

At this point, you should have a directory structure that looks similar to this:

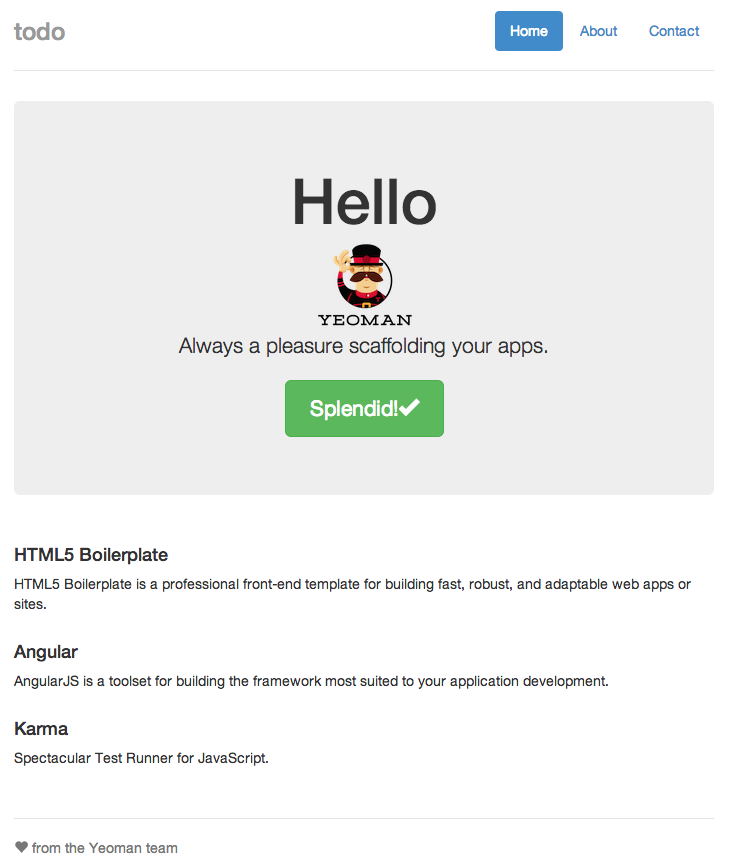


## Run the web app

3. Since yo integrates with bower & grunt, we can run the web app right out of the box. Do so with the command grunt serve.

As part of scaffolding the initial web app, yo defines a custom task called serve that builds & runs your application in a new browser window with LiveReload so that edits are immediately reflected in the browser.

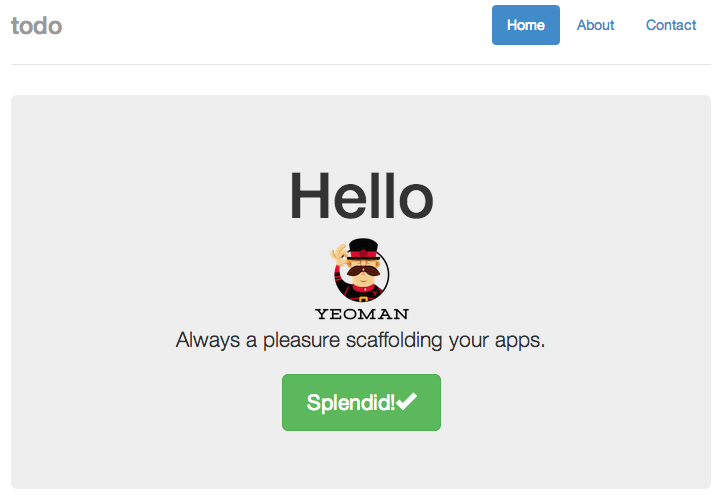
You should see a new browser window pop up with the address http://127.0.0.1:9000/#/ with a view that looks like this:



## Make a live edit

Now, let's see how editing is reflected immediately via LiveReload.

4. Open the file todo/app/views/main.html and change the text in it from <h1>'Allo, 'Allo!</h1> to <h1>Hello</h1>. You should see the new content in the browser without having to refresh the page:



## Use model from controller in view

The page above is primarily statically rendered. Let's change this by using the model provided by the controller in the view.

5. Open the file todo/app/scripts/controllers/main.js and notice that the controller is populating the $scope with a property called awesomeThings with an array of three strings.

Angular routes requests to **controllers**, which produce **models**, which are then available to **views**. A controller places data to be rendered by a view by adding properties to the provided $scope service, given in a controller's constructor function.

For example, in app/app.js, you see the route for the root URL:

$routeProvider

.when('/', {

templateUrl: 'views/main.html',

controller: 'MainCtrl'

})

6. Update the view app/views/main.html to use the model that's produced by controller 'MainCtrl' by changing the content to nothing more than

<div class="container">

<h2>Awesome Things</h2>

<ul>

<li ng-repeat="thing in awesomeThings">{{thing}}</li>

</ul>

</div>

The view should be updated immediately to reflect the new view, which now dynamically renders the model placed in $scope by the controller. When you see this view, move on to the next step.

## Create a new single page application

7. Now, use the scaffolding offered by yo to create a new page ("single page application"), route, controller & view by opening a new terminal, changing to your todo directory, then issuing the command yo angular:route square. The artifacts created are listed below.

First, the route: app/app.js has been updated with the new route

.when('/square', {

templateUrl: 'views/square.html',

controller: 'SquareCtrl'

})

Next, the initial controller implementation in app/scripts/controllers/square.js simply returns some sample data (the same as controller 'MainCtrl'):

angular.module('todoApp')

.controller('SquareCtrl', function ($scope) {

$scope.awesomeThings = [

'HTML5 Boilerplate',

'AngularJS',

'Karma'

];

});

Last, the default view in app/views/square.html is simply

<p>This is the square view.</p>

If you now visit http://127.0.0.1:9000/#/square, you'll see the simple view.

## Update the controller and view to render squares of a few numbers

Now, we're going to update the controller and view so that it renders the squares of a few numbers, but first, we need to abide by good software development principles and ensure that we're testing our controller. Fortunately, grunt provides us with automatic unit and end-to-end testing capabilities.

### Install unit test modules

Before our tests will execute, make sure to install the test modules that we'll need and include them in our development environment's dependencies.

8. Kill the grunt process (via Ctrl-C), then execute npm install karma-jasmine --save-dev and npm install karma-chrome-launcher --save-dev. This will add these modules to the file todo/package.json.

9. Restart the server via grunt serve, then point the browser at http://127.0.0.1:9000/#/square. You should see the simple square view that you did before.

### Write a unit test for the controller

10. Open file todo/spec/controllers/square.js, the file that holds unit tests for the controller 'SquareCtrl', and update the test's it function to be the following:

it('should attach a list of squares of the form { val: n, squared: n\*n } where n is from 1 to 6 to the scope', function () {

expect(scope.squares.length).toBe(6);

[1, 2, 3, 4, 5, 6].forEach(function (elem) {

var obj = scope.squares[elem - 1];

expect(obj).toEqual({

val : elem,

squared : elem \* elem

});

});

});

This places the expectation on our controller to place the expected model into scope.

Once you save this file, you should see an error from grunt, stating something similar to the following:

Chrome 34.0.1847 (Mac OS X 10.9.2) Controller: SquareCtrl should attach a list of squares of the form { val: n, squared: n\*n } where n is from 1 to 6 to the scope FAILED

Expected 0 to be 6.

…

You're seeing this error because our generated controller doesn't meet the specification that we just declared above.

You may need to touch the test file, todo/test/spec/controllers/square.js, in order to get grunt to rerun the tests.

11. Update the controller in app/scripts/controllers/square.js to place an array of objects with two properties each: val, containing a number, and squared, containing the square of the value. Since you're probably as lazy as the author of this lab is and writing code to do this is not the focus of the lab, here's the code.

angular.module('todoApp')

.controller('SquareCtrl', function ($scope) {

$scope.squares = [];

var vals = [1, 2, 3, 4, 5, 6];

vals.forEach(function onEach(it) {

$scope.squares.push({

val: it,

squared: it \* it

});

});

});

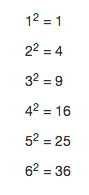
Now, you should see a line similar to the following in your terminal, indicating successful testing (again, you may need to touch the test file again):

Chrome 34.0.1847 (Mac OS X 10.9.2): Executed 2 of 2 SUCCESS (0.027 secs / 0.025 secs)

12. Update the view app/views/square.html to render the values of the object at the key 'squares' in the model (which happens to be an array). Again, here's the code:

<p ng-repeat="sq in squares">{{sq.val}}<sup>2</sup> = {{sq.squared}}</p>

Now, view http://127.0.0.1:9000/#/square in a browser. You should see something similar to this:



It is important to note that the model forms the contract between the controller and the view. The controller is responsible for populating the model, and the view must know the shape of the model so that it can render it properly. Any changes to the view that require new data to be added to the model must be accommodated by the controller so that the controller provides that data.

When you see the expected output, you have completed the lab!