Intro to AngularJS

Presented by Ben Denham

Thanks to Jen Zajac

Administrivia

- Bathrooms
- Fire exits
- Morning tea

What is AngularJS?

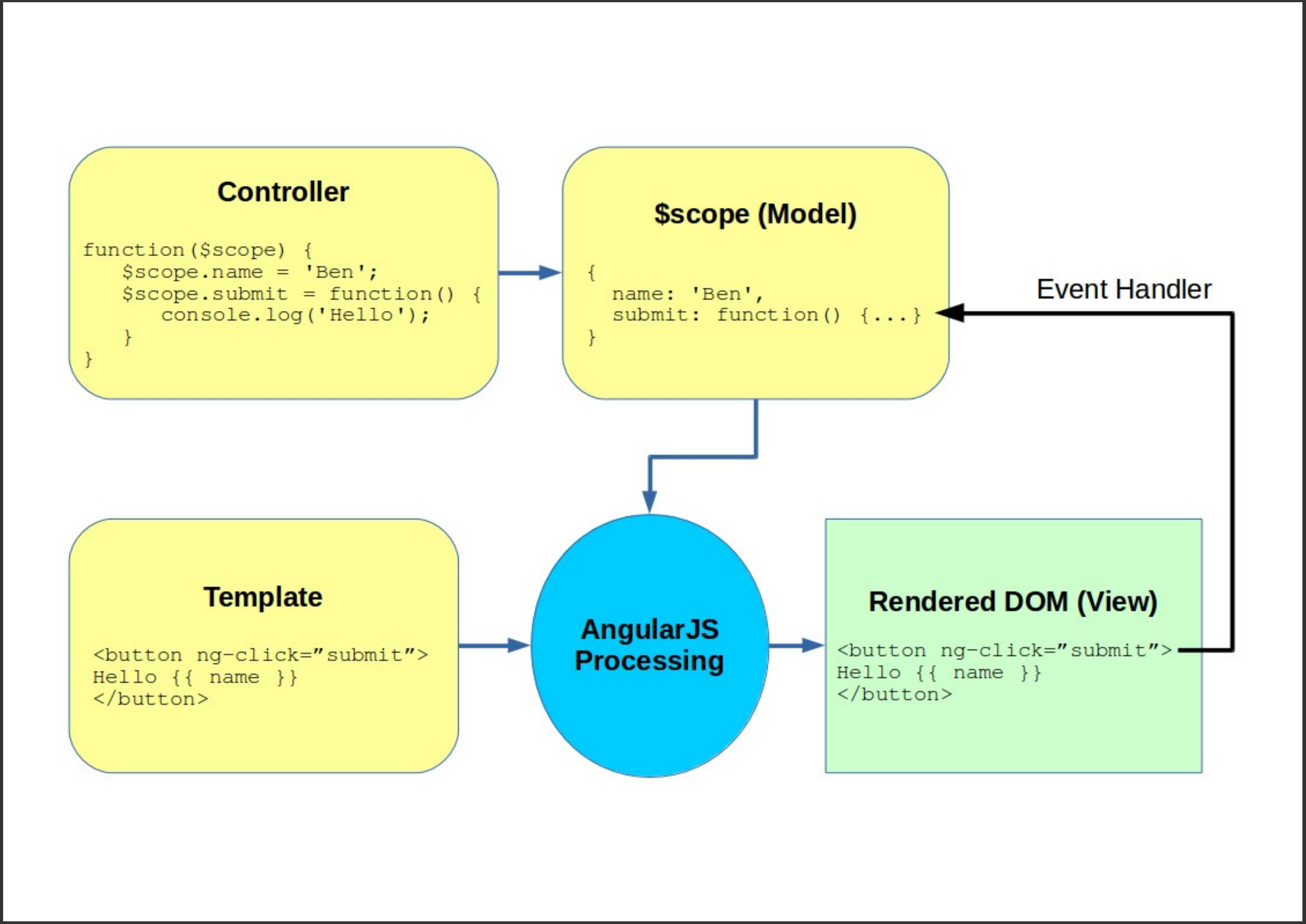
What benefits does it give me?

Two way data binding

"any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view"

MVC
(ish)

MVC in AngularJS



```
function($scope) {
   $scope.name = 'Ben';
   $scope.submit = function() {
      console.log('Hello');
   }
}
```

```
{
  name: 'Ben',
  submit: function() {...}
}
```

```
<button ng-click="submit">
Hello {{ name }}
</button>
```

```
<button ng-click="submit">
Hello {{ name }}
</button>
```

Tests!

Both e2e and Unit

What scenarios would it be a good fit for?

How does it work?

Self contained modules

http://docs.angularjs.org/guide/module

How to divide up your code

Templates

```
<div ng-app="phonecatApp">

    <div ng-controller="myController">
        <h2>{{ myVariable | lowercase }}</h2>
    </div>

</div>
```

Filters?

```
{{ myVariable | lowercase }}
```

Filter

Directives?

```
<button ng-click="submit"></button>
```

Directives

```
<date-picker></date-picker>
```

Controllers & Scope

http://docs.angularjs.org/guide/scope

A cut down version of
jQuery is used under the hood

Services

Useful features

let's look at an actual project

Simple structure

```
app
├── css
│   └── app.css
├── img
├── index.html
├── js
│   ├── app.js
│   ├── controllers.js
│   ├── directives.js
│   ├── filters.js
│   └── services.js
├── lib
│   └── angular
│       └── <angular files>
└── partials
    ├── partial1.html
    └── partial2.html
```

More sophisticated structure

```
app
├── app.css
├── app.js
├── index.html
├── components
│   └── component
│       ├── component.js
│       ├── component_test.js
│       ├── component-filter.js
│       └── component-filter_test.js
├── view
│   ├── view.js
│   ├── view_test.js
│   └── view.html
└── another-view
    ├── another-view.js
    ├── another-view_test.js
    └── another-view.html
```

Open the Demo Application

Open a terminal: `Ctrl-Alt-t`

```
cd intro-to-angular ↩
atom . ↩
npm start ↩
```

↩ = press enter here!

Routes

http://localhost:8000/app/index.html#/view1

/view1/view1.js

```
config(['$routeProvider', function($routeProvider) {
    $routeProvider.when('/view1', {
        templateUrl: 'view1/view1.html',
        controller: 'View1Ctrl'
    });
}])
```

/app.js

```
config(['$routeProvider', function($routeProvider) {
    $routeProvider.otherwise({redirectTo: '/view1'});
}]);
```

Hello, World

view1/view1.js

```
.controller('View1Ctrl', ['$scope', function($scope) {
    $scope.test = 'world';
}]);
```

view1/view1.html

```
<p>Hello {{ test }}.</p>
```

HTTP Requests

Create a file called articles.json in your /app/ directory and paste the content of http://pastebin.com/raw.php?i=NgpHyUsJ in.
Modify view1/view1.js to add the '$http' service:

```
.controller('View1Ctrl', ['$scope', '$http',
  function($scope, $http) {
    $http.get('articles.json').success(function(data) {
        $scope.articles = data;
    });
  });
```

view1/view1.html

```
<pre>
    {{ articles | json }}
</pre>
```

Restful API service

new file – components/articles/articles-services.js

```
'use strict';
angular.module('myApp.articles.articles-services', [])
.factory('Articles', ['$resource', function($resource){
    return $resource('articles.json', {}, {
        query: {method:'GET', params:{}, isArray:true}
    });
}]);
```

new file – components/articles/articles.js

```
'use strict';
angular.module('myApp.articles', [
  'myApp.articles.articles-services'
]);
```

Restful API service continued

index.html

```
<script src="bower_components/angular-resource/angular-
resource.js"></script>
<script src="components/articles/articles.js"></script>
<script src="components/articles/articles-
services.js"></script>
```

app.js

```
angular.module('myApp', [
  'ngRoute',
  'ngResource',
  'myApp.articles'
...
```

view1/view1.js

```
controller('View1Ctrl', ['$scope', 'Articles',
    function($scope, Articles) {
        $scope.articles = Articles.query();
```

Writing a unit test

new file – components/articles/articles-services_test.js

```
'use strict';

describe('myApp.articles module', function() {
  beforeEach(module('myApp.articles', 'ngResource'));

  describe('articles service', function() {

    it('should make the Articles factory available',
      inject(function(Articles, $resource) {
        expect(Articles).toBeDefined();
    }));

  });
});
```

../karma.conf.js – add to files array

```
'app/bower_components/angular-resource/angular-resource.js',
```

Writing a unit test

run unit tests (in a new terminal tab, from root of the project)

```
npm test ↩
```

↩ = press enter here!

View unit test

view1/view1.test

```
  beforeEach(module('myApp.view1', 'myApp.articles', 
'ngResource'));

  describe('view1 controller', function(){

    var scope, ctrl;
    beforeEach(inject(function($rootScope, $controller) {
        scope = $rootScope.$new();
        ctrl = $controller('View1Ctrl', {$scope: scope});
    }));

    it('should have a scope', inject(function() {
      expect(ctrl).toBeDefined();
    }));

  });
```

Unit test with a mock

view1/view1.test

```
    var $httpBackend, scope, ctrl;
    beforeEach(inject(function(_$httpBackend_, $rootScope,
$controller) {
        $httpBackend = _$httpBackend_;
        $httpBackend.expectGET('articles.json').
            respond([
               {title: 'Test Article 1'},
               {title: 'Test Article 2'}
            ]);
        scope = $rootScope.$new();
        ctrl = $controller('View1Ctrl', {$scope: scope});
    }));

    it('should create an "articles" model from xhr data',
        inject(function() {
            expect(scope.articles.length).toBe(0);
            $httpBackend.flush();
            expect(scope.articles.length).toBe(2);
    }));
```

getting stuck in

Sidenote:

Use the docs!

http://docs.angularjs.org/api/

Built-in directives

ng-repeat

view1/view1.html

```
<ul>
    <li ng-repeat='article in articles'>
         “{{ article.title }}” by
         {{ article.authors }}
    </li>
</ul>
```

ng-show

view1/view1.html

```
<ul>
    <li
       ng-repeat='article in articles'
       ng-show='!article.archived'
    >
        “{{ article.title }}” by
        {{ article.authors }}
    </li>
</ul>
```

ng-class

view1/view1.html

```
<ul>
    <li
       ng-repeat='article in articles'
       ng-class='{archived:article.archived}'
    >
        “{{ article.title }}” by
        {{ article.authors }}
    </li>
</ul>
```

app.css

```
.archived {
  color:gray;
}
```

ng-src

view1/view1.html

```
<ul class='articles'>
    <li
       ng-repeat='article in articles'
       ng-class='{archived:article.archived}'
    >
         <img ng-src="{{ article.icon }}"><br>
         “{{ article.title }}” by
         {{ article.authors }}
    </li>
</ul>
```

app.css

```
.articles li {
  list-style-type:none;
  margin-bottom:20px;
}
```

ng-click

view1/view1.html

```
<p><button ng-click="hideAuthors = true">Hide
authors</button><p>

<ul class='articles'>
    <li
       ng-repeat='article in articles'
       ng-class='{archived:article.archived}'
    >
         <img ng-src="{{ article.icon }}"><br>
         “{{ article.title }}”
         <span ng-show='!hideAuthors'>
            by {{ article.authors }}
         </span>
    </li>
</ul>
```

A little more styling...

app.css

```
.articles {
  padding-left:0;
}

body {
    padding:20px;
}
```

Built-in filters

limitTo

view1/view1.html

```
<ul class='articles'>
    <li
       ng-repeat='article in articles | limitTo:5'
       ng-class='{archived:article.archived}'
    >
         <img ng-src="{{ article.icon }}"><br>
         “{{ article.title }}”
         <span ng-show='!hideAuthors'>
            by {{ article.authors }}
         </span>
    </li>
</ul>
```

filter

view1/view1.html

```
<p><label>Search:
   <input type="search" value="" ng-model="userInput">
</label></p>

<p>Searched for: {{ userInput }}</p>

<p ng-show="(articles | filter:userInput).length ==
0"><em>No articles found.</em></p>

<ul class='articles'>
    <li
       ng-repeat='article in articles |
                  filter:userInput'
```

filter with more specificity

view1/view1.html

```
<p><label>Search:
    <input type="search" value="" ng-
model="userInput.authors">
</label></p>

<p>Searched for: {{ userInput.authors }}</p>
```

Writing an e2e test

../e2e-tests/scenarios.js – replace “should render view1 when user navigates to /view1”

```
it('reduces the list of articles the user enters a search term',
    function() {

expect(element.all(by.css('.articles li')).count()).toBeGreaterThan(1);
element(by.css('[ng-model="userInput.authors"]')).sendKeys('Holmes');
expect(element.all(by.css('.articles li')).count()).toBe(1);

});
```

run unit tests (in a new terminal tab, from root of the project)

```
npm run protractor ↩
```

orderBy

view1/view1.html

```
<p>
   <button ng-click="alphabetical = !alphabetical">Change
   sort order</button>
<p>
<ul class='articles'>
    <li ng-repeat='article in articles |
                   orderBy:sortOrder(alphabetical)'
```

view1/view1.js

```
$scope.alphabetical = true;
$scope.sortOrder =  function(isAlphabetical) {
    if (isAlphabetical) {
        return ['title', 'published'];
    }
    return ['-title', '-published'];
};
```

Writing our own filter

Separating the Author names with commas

view1/view1.html

```
<span ng-show="!hideAuthors">
   by {{ article.authors | authorList }}
</span>
```

new file – components/articles/articles-filters.js

```
'use strict';

angular.module('myApp.articles.articles-filters', [])

.filter('authorList', [function(){
    return function(array) {
        return array.join(', ');
    }
}]);
```

Separating the Author names with commas, continued

components/articles/articles.js

```
...
  'myApp.articles.articles-filters'
```

index.html

```
<script src="components/articles/articles-filters.js"></script>
```

Writing our own directive

Making our article HTML into a directive

index.html

```
<script src="components/articles/articles-directives.js"></script>
```

view1/view1.html

```
    <li
       ng-repeat='article in articles |
orderBy:sortOrder(alphabetical) | filter:userInput'
       ng-class='{archived:article.archived}'
    >
        <article>
    </li>
```

components/articles/article.html

```
<img ng-src="{{ article.icon }}"><br>“{{ article.title }}”
<span ng-show='!hideAuthors'>by {{ article.authors |
authorList }}</span>
```

Making our article HTML into a directive, continued

components/articles/articles.js

```
...
  'myApp.articles.articles-directives'
```

components/articles/articles-directives.js

```
'use strict';

angular.module('myApp.articles.articles-directives', [])
.directive('article', [function() {
  return {
    restrict: 'E',
    templateUrl: 'components/articles/article.html'
  };
}]);
```

Other built in stuff of interest

Services: location, window, q, animate, sanitize
Filters: date, currency, lowercase
Directives: paste, switch, angularUI

...loads more, read the docs!

Other resources

Official tutorial

Todo MVC

Coming from a jQuery background

Very simple todo app, step by step

Project structure best practice

Intro to AngularJS

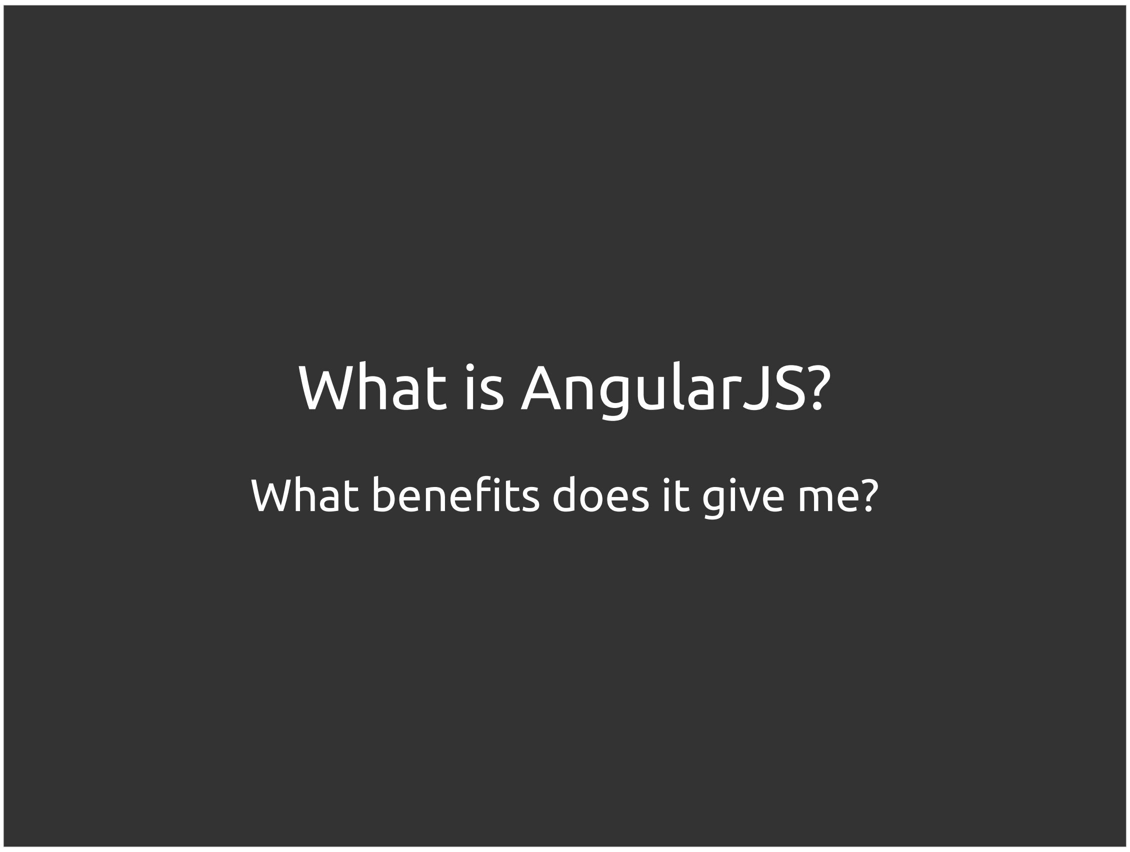
Presented by Ben Denham

Thanks to Jen Zajac

I do suggest following along with the tutorial on the offical site afterwards in addition to taking this training.

Administrivia

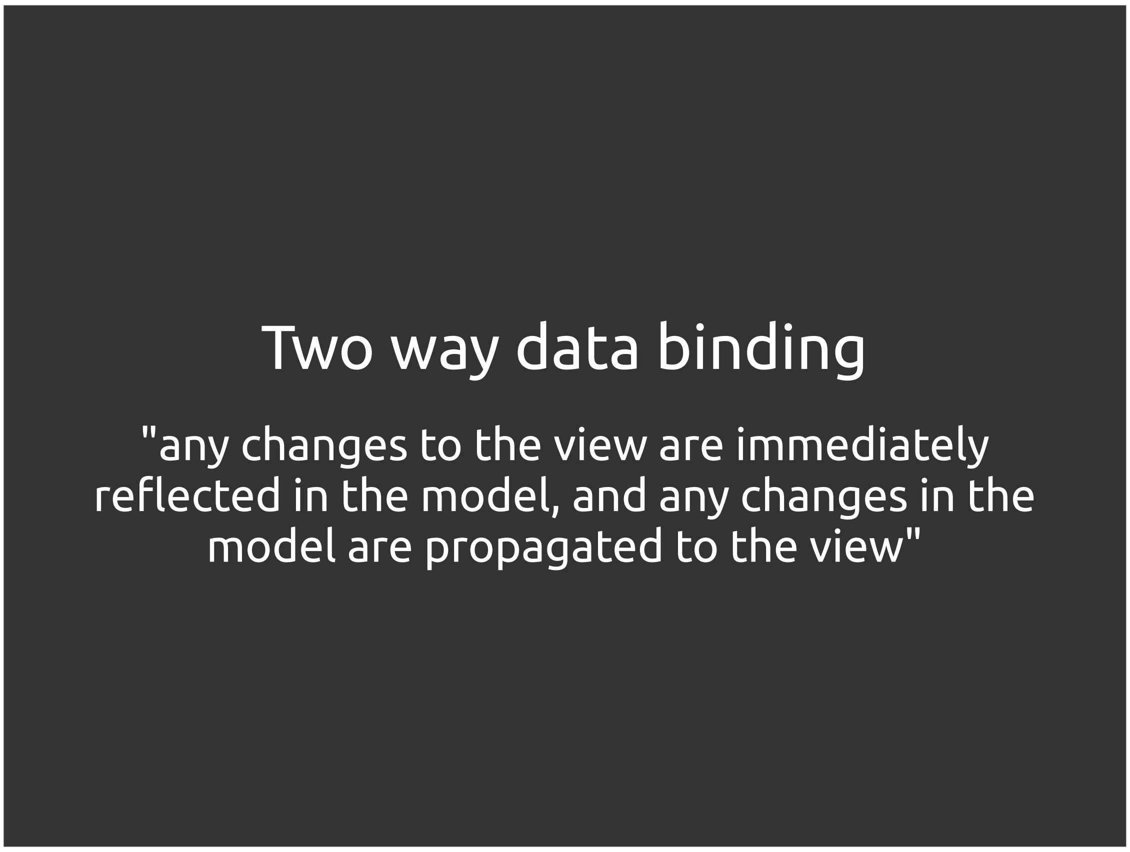
- Bathrooms
- Fire exits
- Morning tea



What is AngularJS?

What benefits does it give me?

See next slides :)



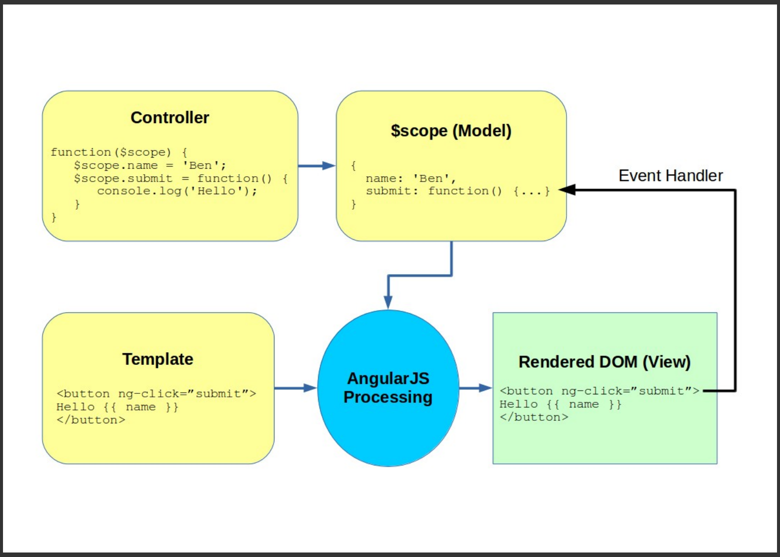
Two way data binding

"any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view"

- as the user interacts with the page, changes flow through immediately
- Editing a textfield updates the variable, editing a variable updates the textfield.

MVC
(ish)

MVC in AngularJS



```
function($scope) {
   $scope.name = 'Ben';
   $scope.submit = function() {
      console.log('Hello');
   }
}
```

```
{
  name: 'Ben',
  submit: function() {...}
}
```

```
<button ng-click="submit">
Hello {{ name }}
</button>
```

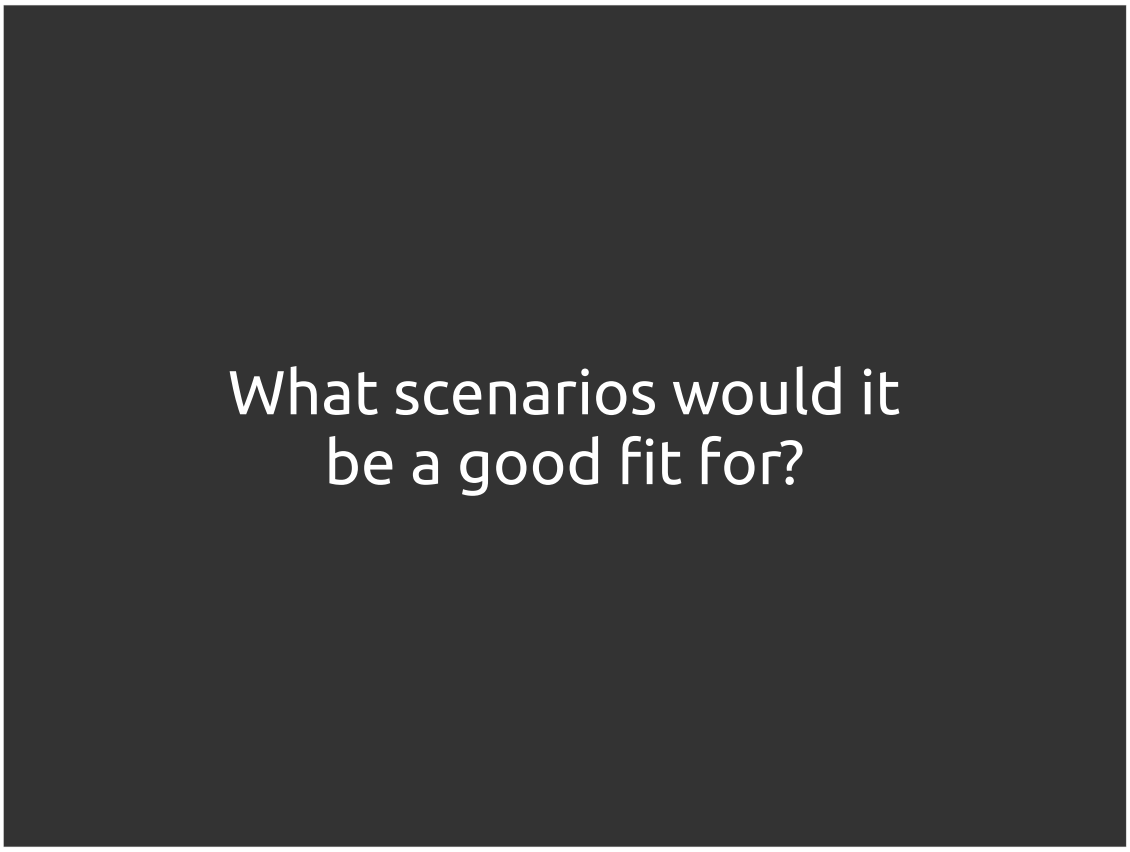
```
<button ng-click="submit">
Hello {{ name }}
</button>
```

- model is any data that is reachable as a property of an angular Scope object
- the view is the DOM loaded and rendered in the browser, after Angular has transformed the DOM based on information in the template, controller and model
- controller is a JavaScript function(type/class) that is used to augment instances of angular Scope. should be unique business logic only, everything else should be a service, directive or filter

Tests!

Both e2e and Unit

- uses node.js based 'Karma' test runner
- can substitute other test runners if you like e.g. I have used Jasmine's HTML based unit test runner
- suggested to use Jasmine for the syntax of the unit tests e.g. expect(thing.length).toBe(3);



What scenarios would it be a good fit for?

- new site, or new section of site
- talking to api
- making page more responsive
- watch out for browser compatibility
    - IE8 and above
- historically not great for animation, but there are animation features now
- not good to have two way binding for pages with LOTS of data e.g. > 500ish rows in a table is starting to get a bit big

How does it work?

See next slides :)

Self contained modules

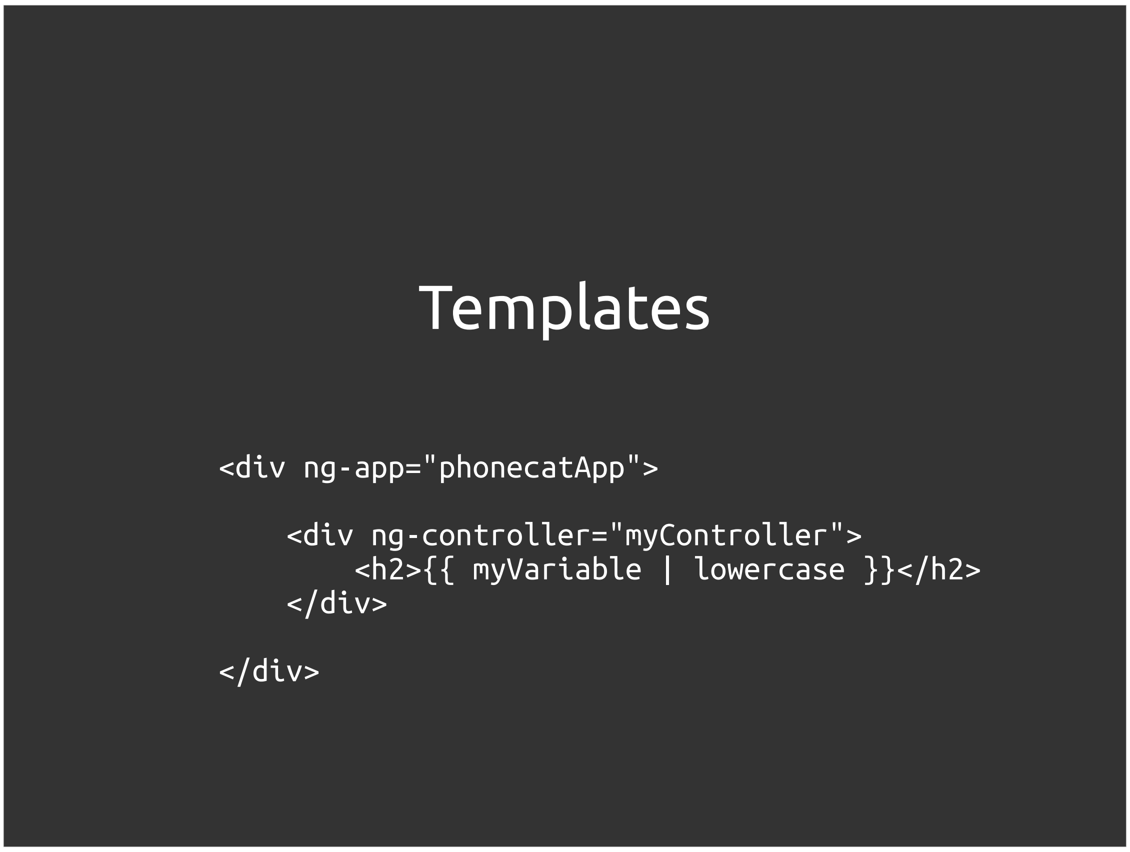
http://docs.angularjs.org/guide/module

- Angular apps don't have a main method
- Uses modules instead, which are written in a declarative fashion (always state exactly what other modules are required)
- **Conventions govern how Angular applications typically have modules set up**
- In unit-testing there is no need to load all modules, which may aid in writing unit-tests
- Additional modules can be loaded in scenario tests, which can override some of the configuration and help end-to-end test the application
- Easy segregation of third party code

How to divide up your code

- used to be: controllers, directives, filters
- app.js is the main 'setup'
- can be one .js file, but conventionally you seperate stuff out
- for big projects, it's better to split things up by functionality

A good approach:
- A module for each feature
- A module for each reusable component (especially directives and filters)
- And an application level module which depends on the above modules and contains any initialization code.

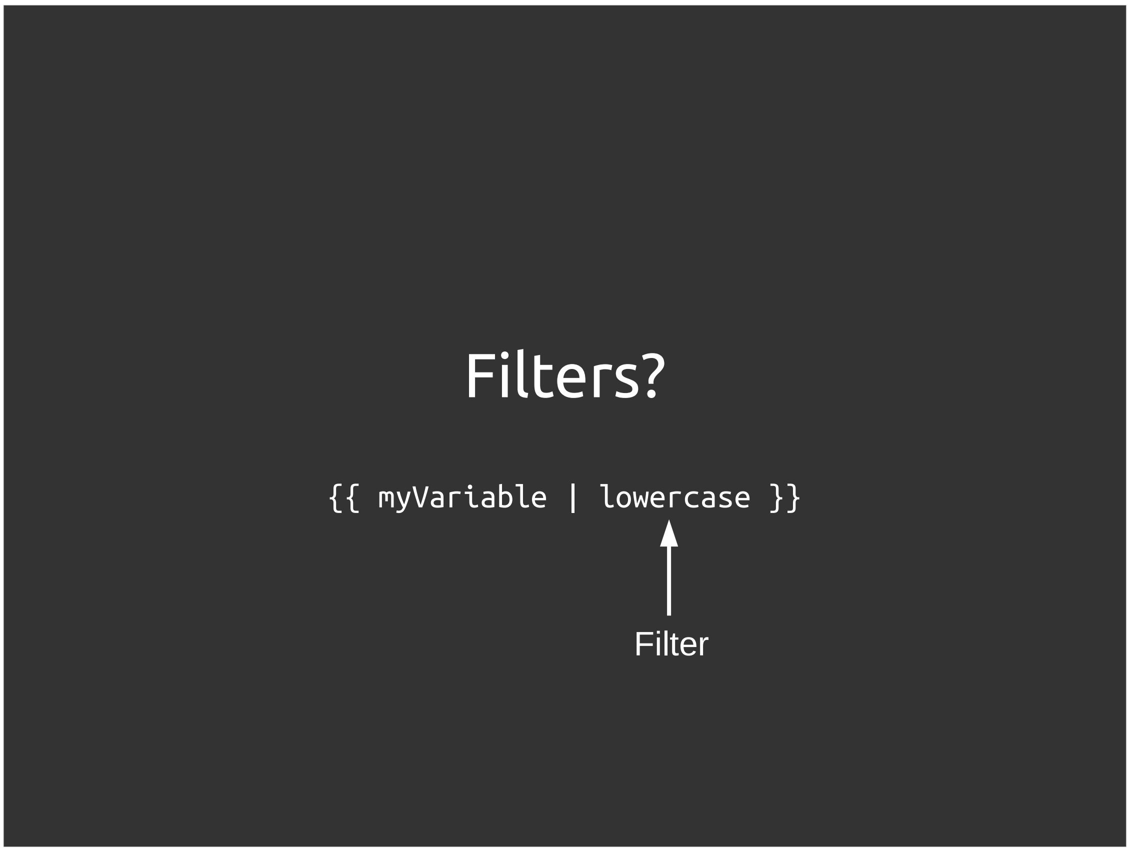


Templates

```
<div ng-app="phonecatApp">

    <div ng-controller="myController">
        <h2>{{ myVariable | lowercase }}</h2>
    </div>

</div>
```

- templates use a mix of HTML, handlebars-style curly braces, and angular attributes (prefixed with ng-

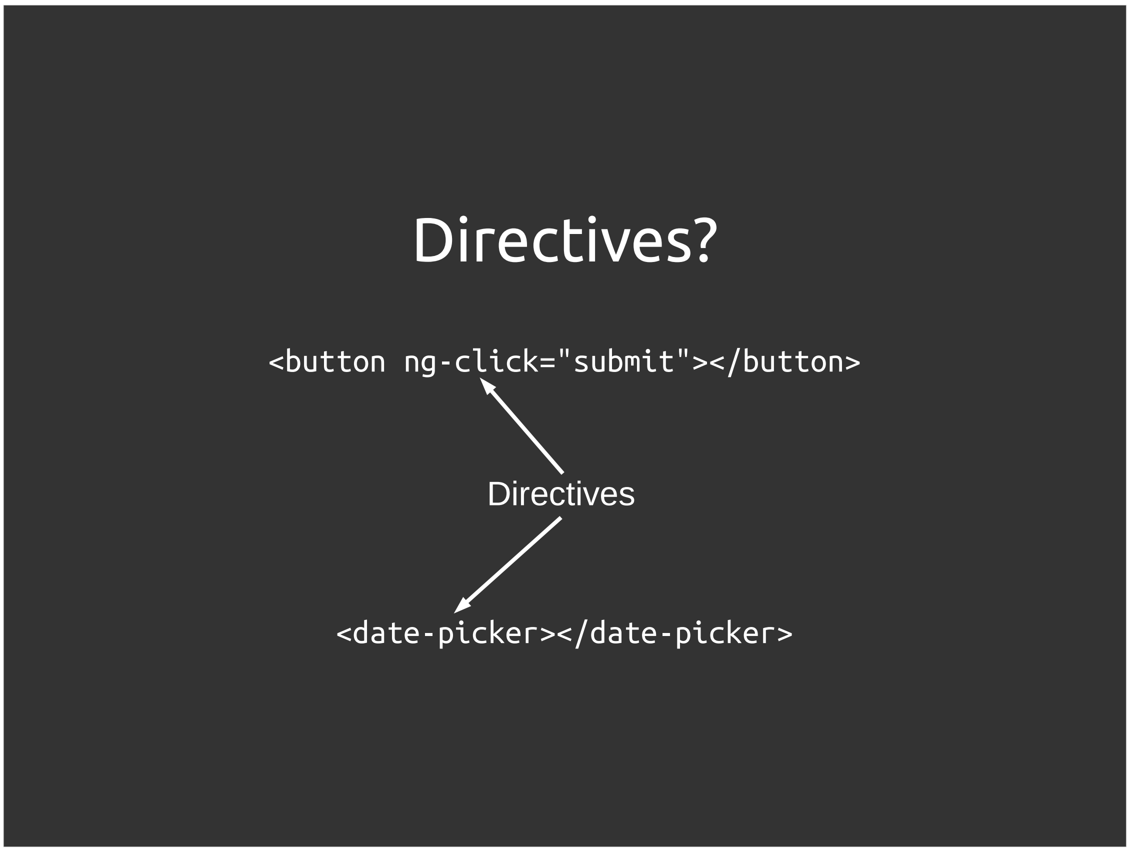
- as a convention, any attributes will be lowercased and use hyphens as seperators, but in the javascript that will be translated to camelCase. E.g. ng-controller would be referrred to in Angular's source code as ngController. This is more relevant once we start writing our own directives



Filters?

```
{{ myVariable | lowercase }}
```

- a filter formats the value of an expression for display to the user
- Filters can be used both in templates and within controllers or other Angular JS code. Angular comes with a number of built in filters, and we can write our own



Directives?

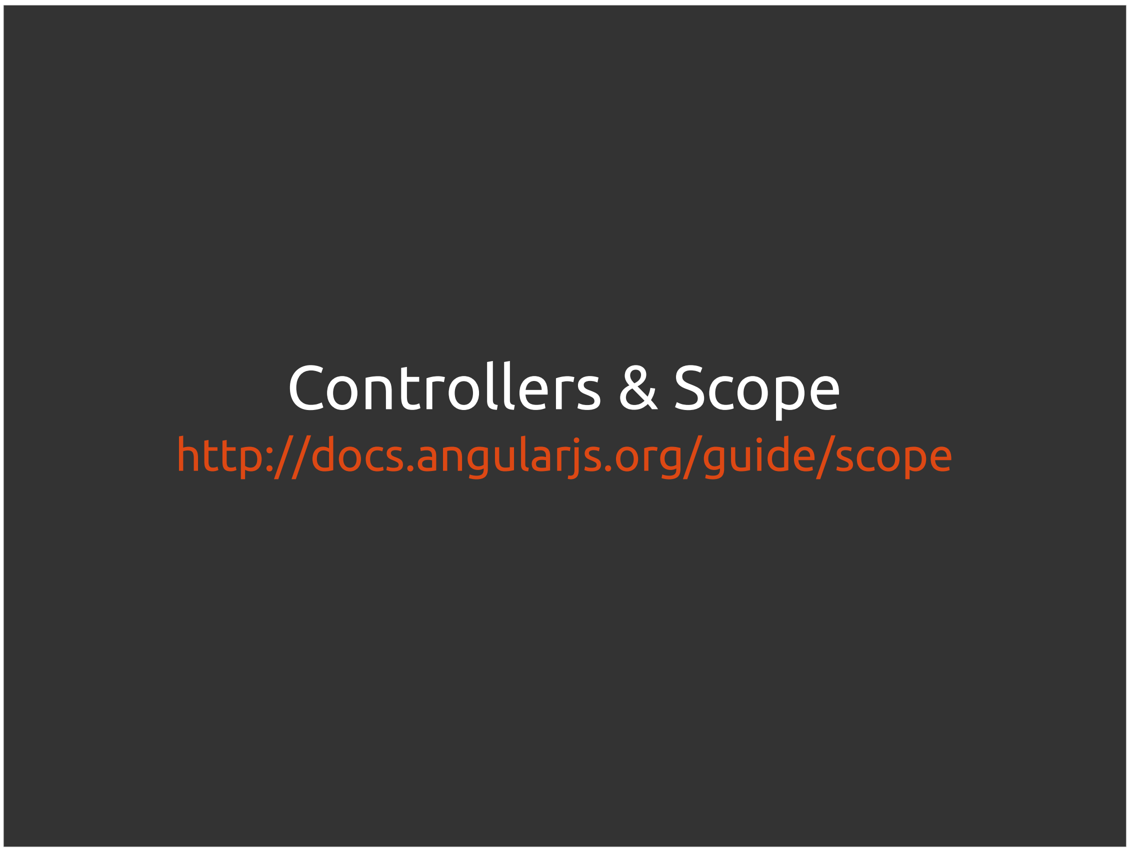
```
<button ng-click="submit"></button>
```

Directives

```
<date-picker></date-picker>
```

- directives are the building blocks for custom DOM elements, or for event handling etc
- To create a date picker with **jQuery** we would first add an input to the HTML, and in the JS we'd call $(element).datePicker(). **Figuring out the realtion between the HTML and the javascript is not straightforwards later.**
- A **directive** for a date picker might look like this: <date-picker></date-picker>. When the template is rendered it will be replaced. **It's more obvious that this is a custom component** (and we can make it more so with naming conventions)

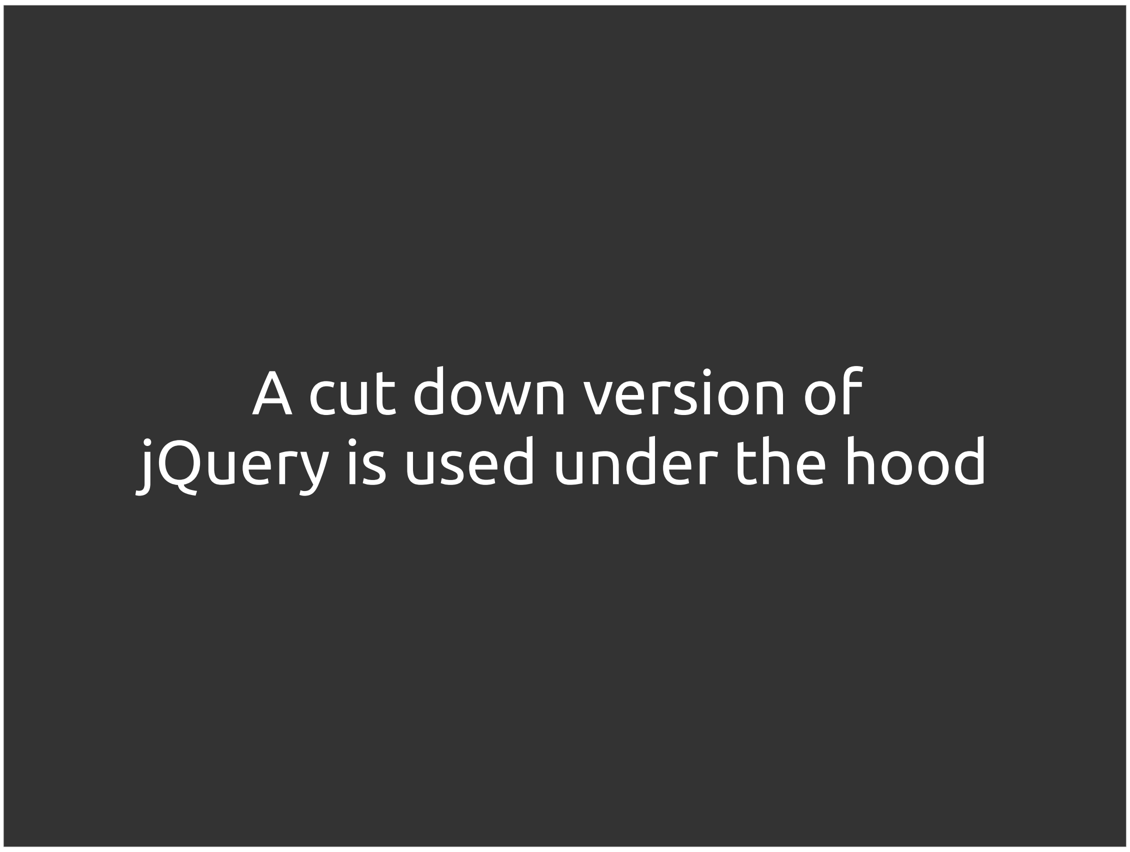
Each directive can have a template to replace the directive element with, and “controller-like” code.



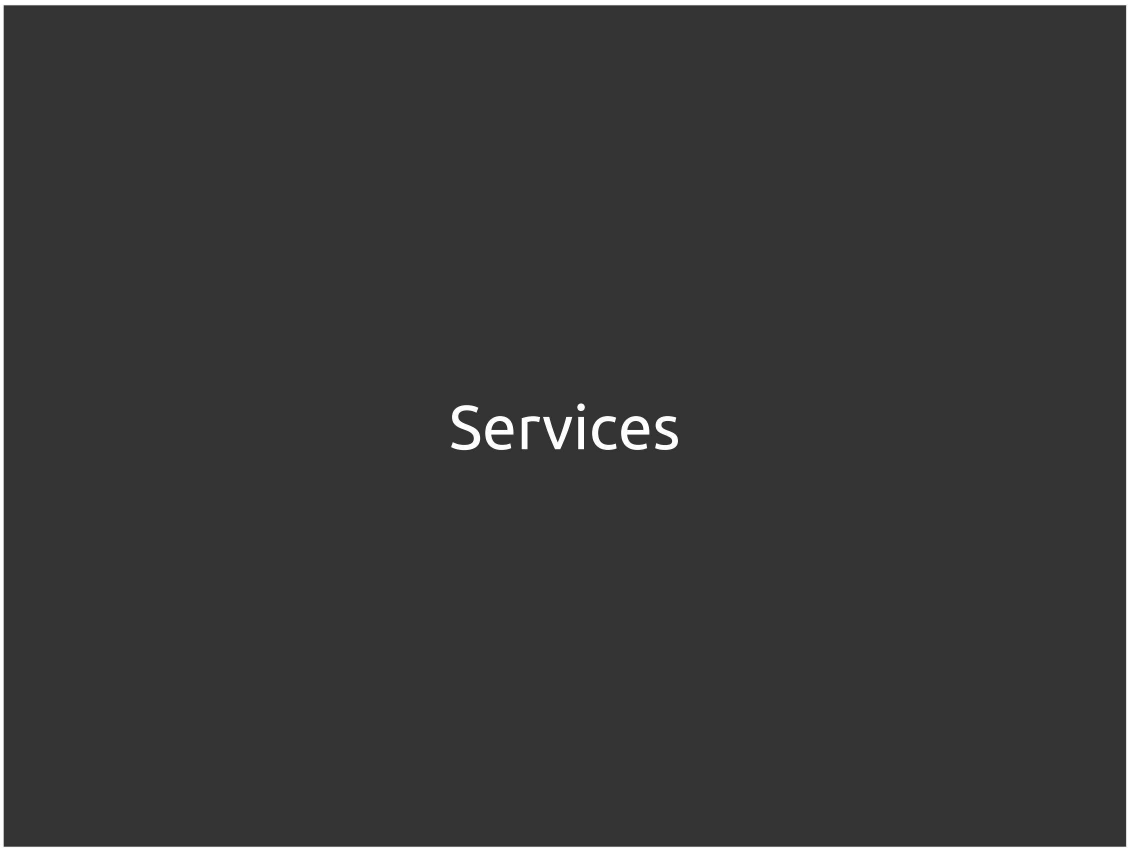
Controllers & Scope

http://docs.angularjs.org/guide/scope

- This is what makes that two way data binding we talked about work
- "Scope is the glue between application controller and the view."
- **scope hierarchies**
- descended from root scope
- scopes get updated automatically via 'watch' code (but can take manual control with $scope.apply())


A cut down version of
jQuery is used under the hood

- jqlite
- subset of jquery selectors are available
- you can include jquery if you want
- e.g. I often include it to use jQuery's $.map
- you SHOULD NOT be using jQuery selectors within
  your controllers


Services

- there is one other 'type' of contruct within Angular, a service.
- Services are a bit harder to define.
- I like to think of them as code that could be in your controller, but that you want to be reusable by other controllers.

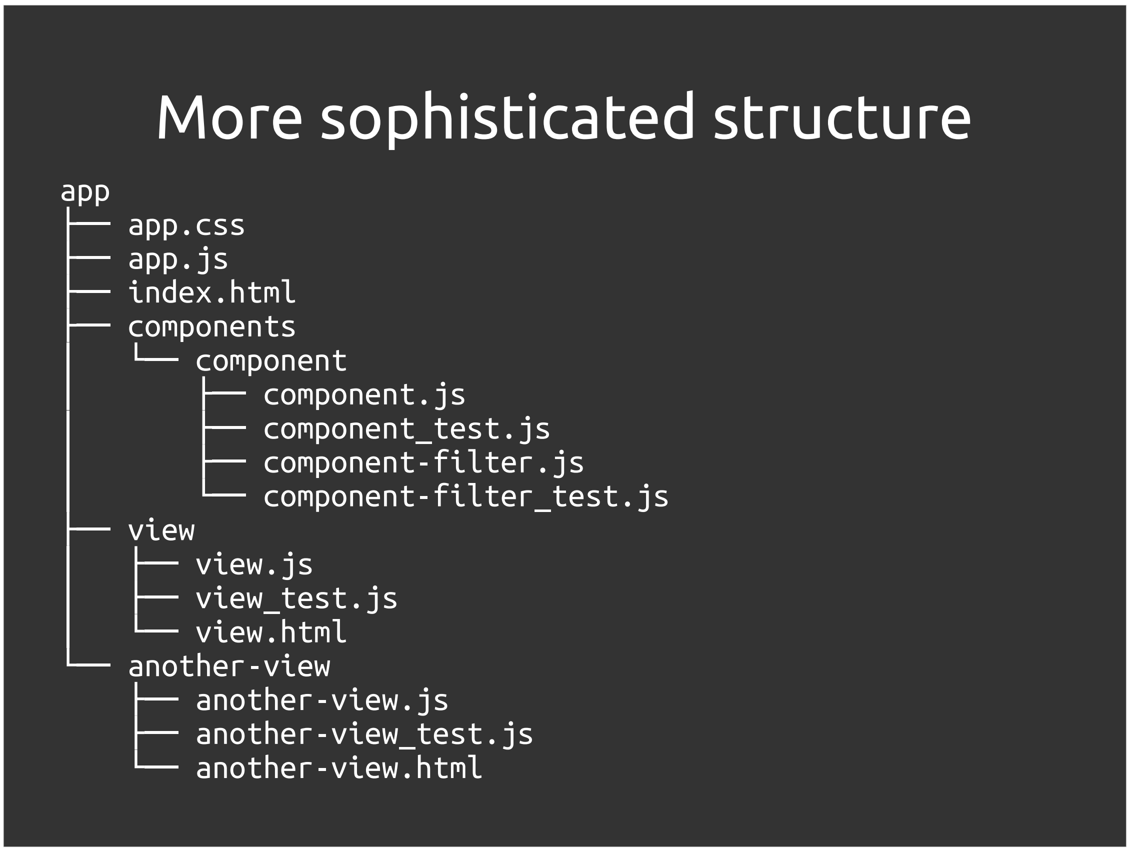
Useful features

- LOTS of built in cool things you can do
- better to show you than to tell you

let's look at an actual project

Simple structure

```
app
├── css
│   └── app.css
├── img
├── index.html
├── js
│   ├── app.js
│   ├── controllers.js
│   ├── directives.js
│   ├── filters.js
│   └── services.js
├── lib
│   └── angular
│       └── <angular files>
└── partials
    ├── partial1.html
    └── partial2.html
```



More sophisticated structure

```
app
├── app.css
├── app.js
├── index.html
├── components
│   └── component
│       ├── component.js
│       ├── component_test.js
│       ├── component-filter.js
│       └── component-filter_test.js
├── view
│   ├── view.js
│   ├── view_test.js
│   └── view.html
└── another-view
    ├── another-view.js
    ├── another-view_test.js
    └── another-view.html
```

The structure of the app we will work on (based on Angular Seed).

We're going to start by looking at what is in view/

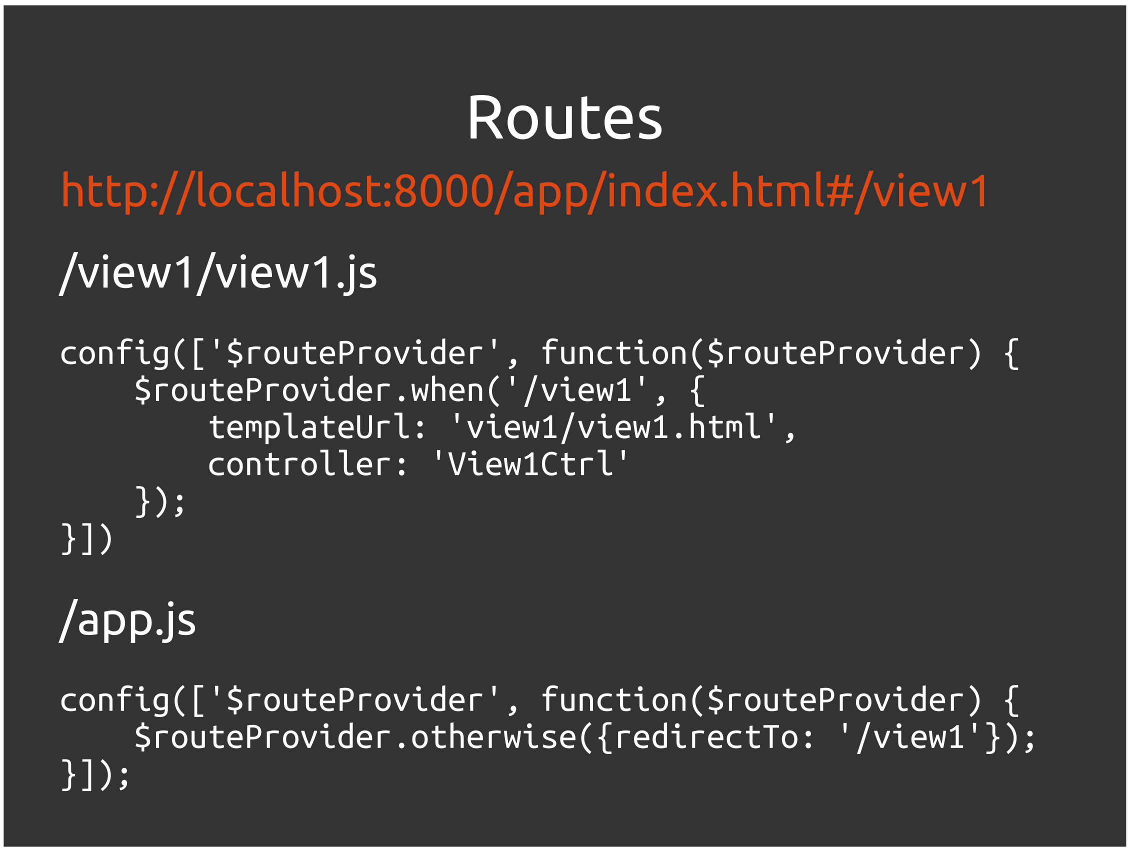
Follow along in view1/ of the repo.

Open the Demo Application

Open a terminal: `Ctrl-Alt-t`

```
cd intro-to-angular ↩
atom . ↩
npm start ↩
```

↩ = press enter here!


Routes

http://localhost:8000/app/index.html#/view1

/view1/view1.js

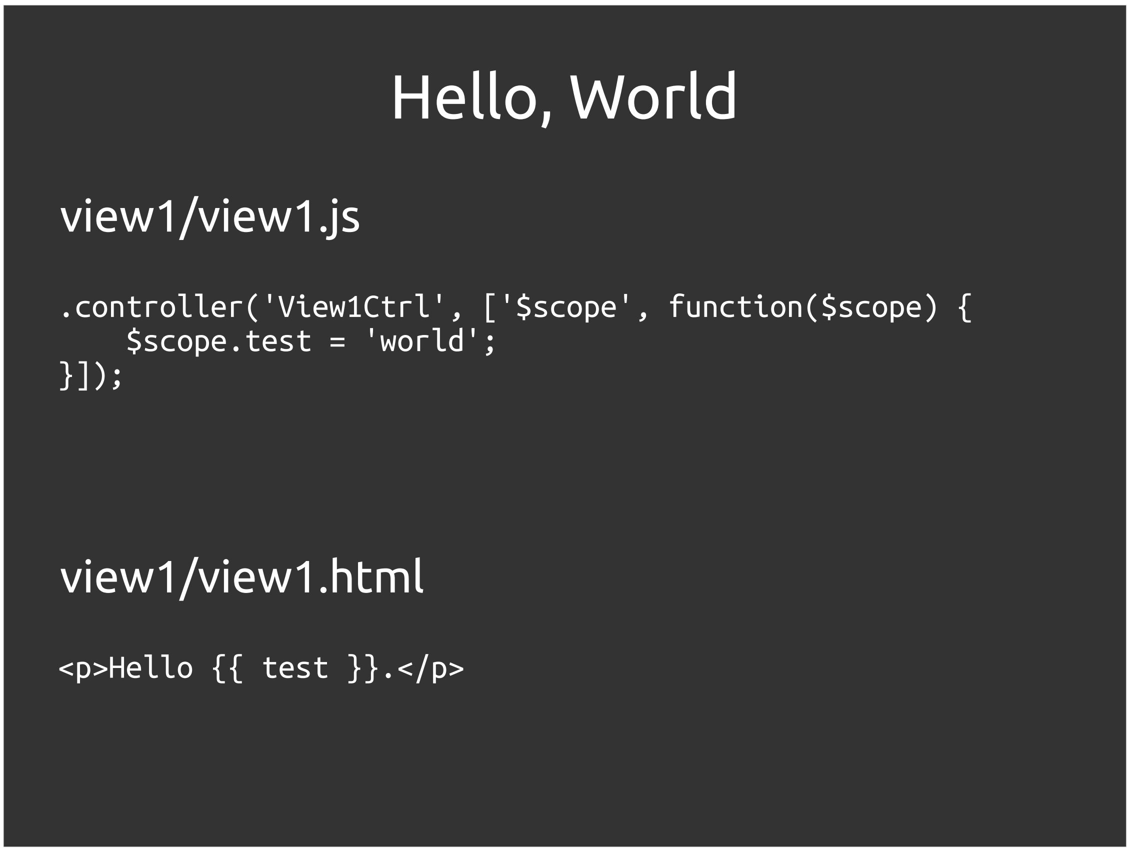
```
config(['$routeProvider', function($routeProvider) {
    $routeProvider.when('/view1', {
        templateUrl: 'view1/view1.html',
        controller: 'View1Ctrl'
    });
}])
```

/app.js

```
config(['$routeProvider', function($routeProvider) {
    $routeProvider.otherwise({redirectTo: '/view1'});
}]);
```

Code from here to slide 30 is already in the repo, so we just walk through it.

- everything after the # fragment is part of the Angular app
- the index.html bit wouldn't usually be there (it's just because the node server we're using isn't very sophisticated)
- we can use HTML5 push state (for browsers that support it) to do away with the # altogether (but it also requires setup of the webserver)
- in our app.js we associate particular routes with particular partial templates
- **we don't have to use routes and partials if we don't want to** - alternatively you could either have a one page app or you can manually associate particular controllers with particular HTML pages whereever you like



Hello, World

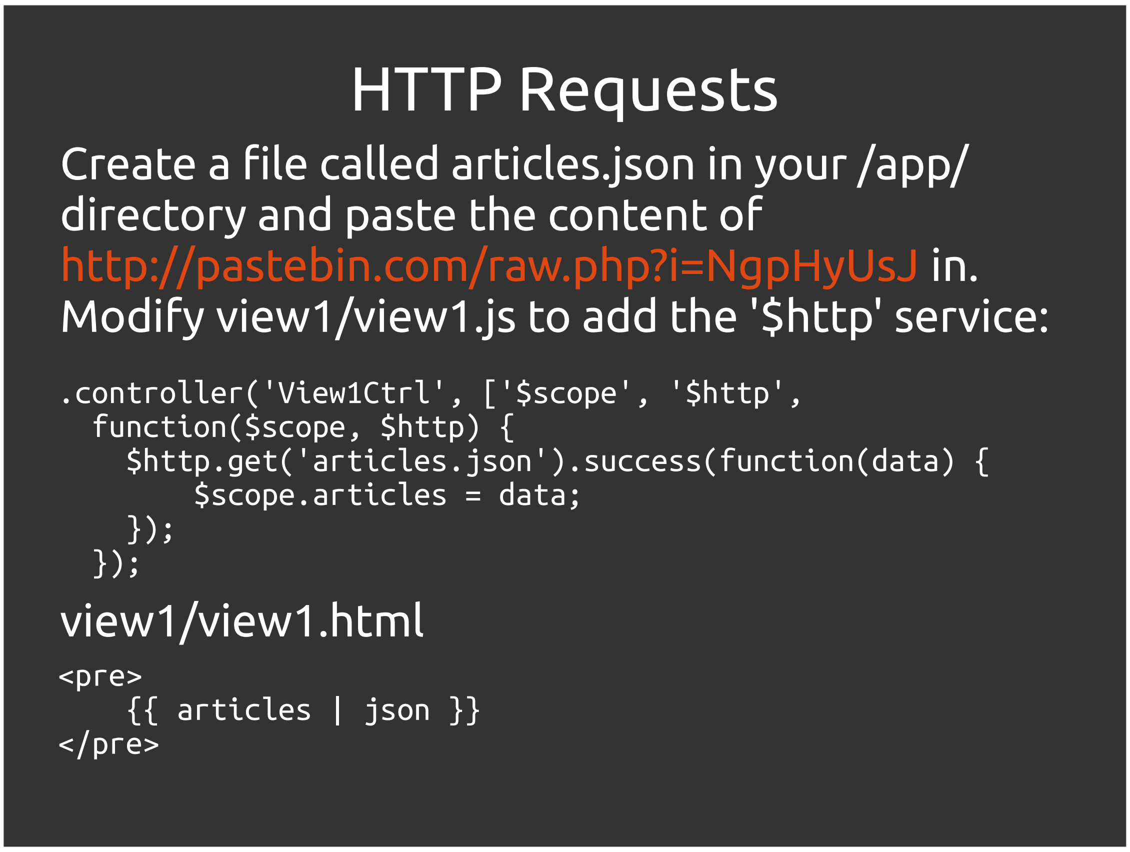
view1/view1.js

```
.controller('View1Ctrl', ['$scope', function($scope) {
    $scope.test = 'world';
}]);
```

view1/view1.html

```
<p>Hello {{ test }}.</p>
```

- modify controllers.js to inject $scope into MyCtrl1
- add a scope variable inside the controller function
- modify partial1.html to output the variable
- **the square brackets prevent later issues with minification renaming variables**



HTTP Requests

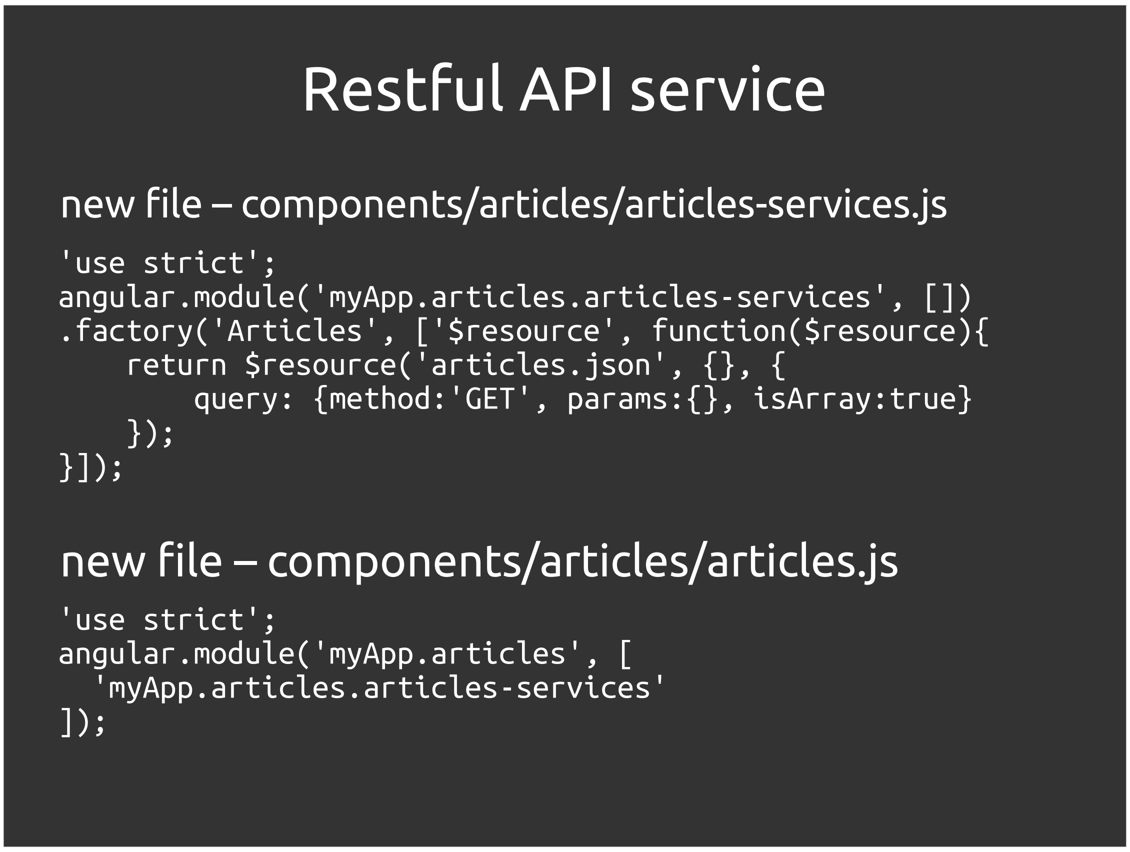
Create a file called articles.json in your /app/ directory and paste the content of http://pastebin.com/raw.php?i=NgpHyUsJ in.
Modify view1/view1.js to add the '$http' service:

```
.controller('View1Ctrl', ['$scope', '$http',
  function($scope, $http) {
    $http.get('articles.json').success(function(data) {
        $scope.articles = data;
    });
  });
```

view1/view1.html

```
<pre>
    {{ articles | json }}
</pre>
```

- watch out for cors
  http://www.html5rocks.com/en/tutorials/cors/
- the `| json` is an example of a filter being used


Restful API service

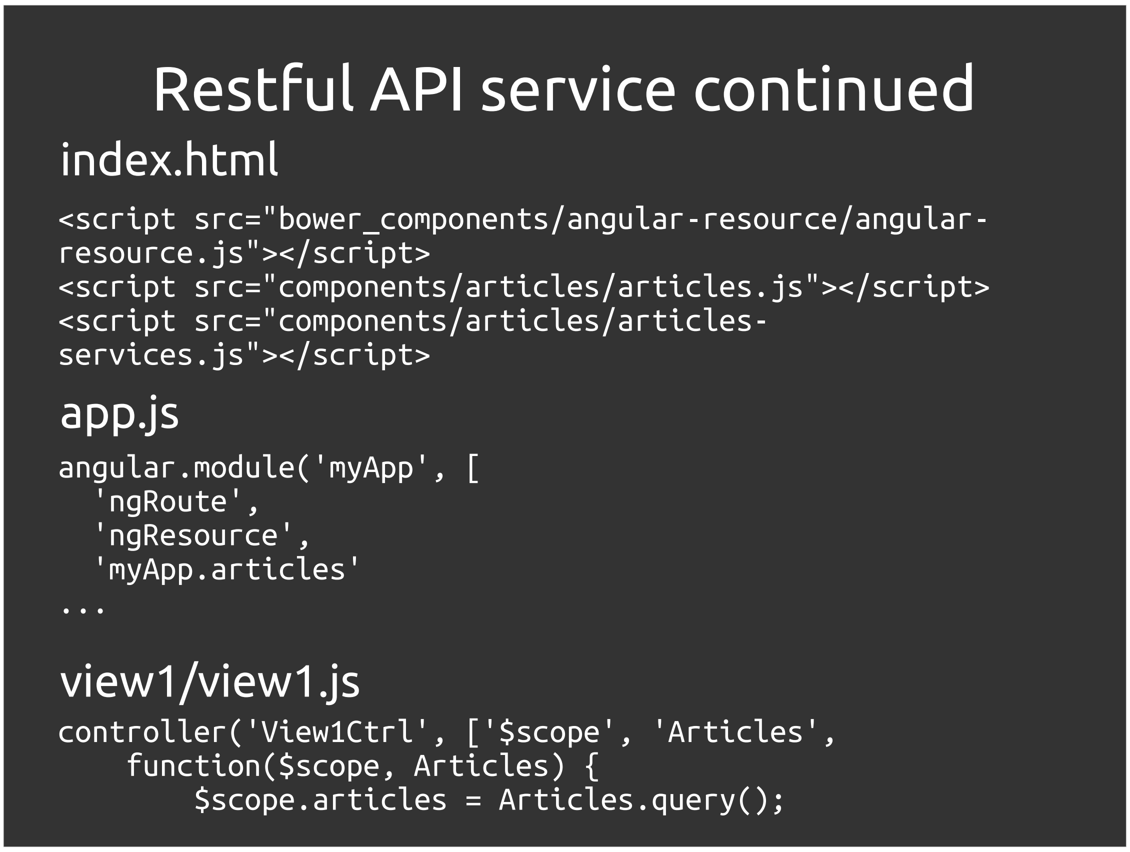
new file – components/articles/articles-services.js

```
'use strict';
angular.module('myApp.articles.articles-services', [])
.factory('Articles', ['$resource', function($resource){
    return $resource('articles.json', {}, {
        query: {method:'GET', params:{}, isArray:true}
    });
}]);
```

new file – components/articles/articles.js

```
'use strict';
angular.module('myApp.articles', [
  'myApp.articles.articles-services'
]);
```

- now we'll swap the low level http service for a proper restful service using ng-resource
- This would really come into it's own when we started doing more than just GET
- Takes all of the http boilerplate away, no more dealing with URLs in the controllers
- Services can be used for more than just this – any code that isn't a directive or filter but that you will need between multiple controllers should be services
- By convention services are capitalised



Restful API service continued

index.html

```
<script src="bower_components/angular-resource/angular-
resource.js"></script>
<script src="components/articles/articles.js"></script>
<script src="components/articles/articles-
services.js"></script>
```

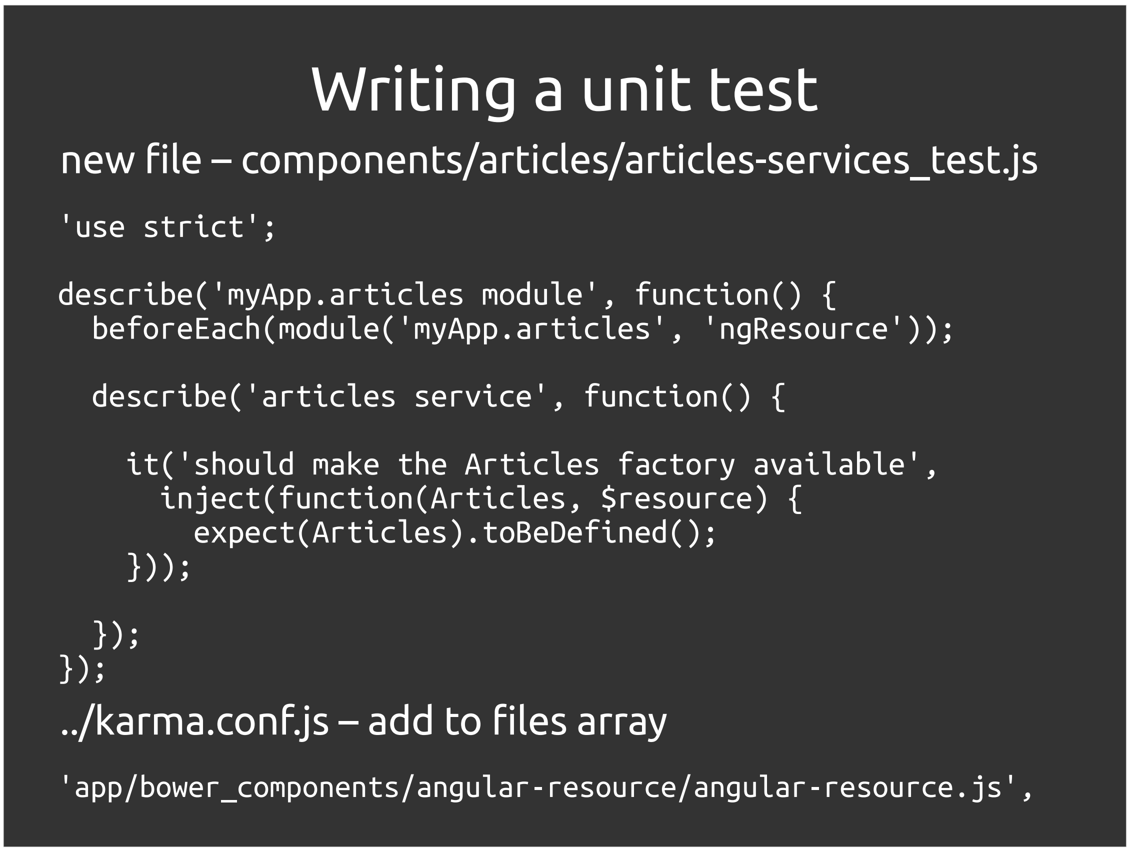
app.js

```
angular.module('myApp', [
  'ngRoute',
  'ngResource',
  'myApp.articles'
...
```

view1/view1.js

```
controller('View1Ctrl', ['$scope', 'Articles',
    function($scope, Articles) {
        $scope.articles = Articles.query();
```

- now we'll swap the low level http service for a proper restful service using ng-resource
- This would really come into it's own when we started doing more than just GET
- Takes all of the http boilerplate away, no more dealing with URLs in the controllers
- Services can be used for more than just this – any code that isn't a directive or filter but that you will need between multiple controllers should be services
- By convention services are capitalised



Writing a unit test

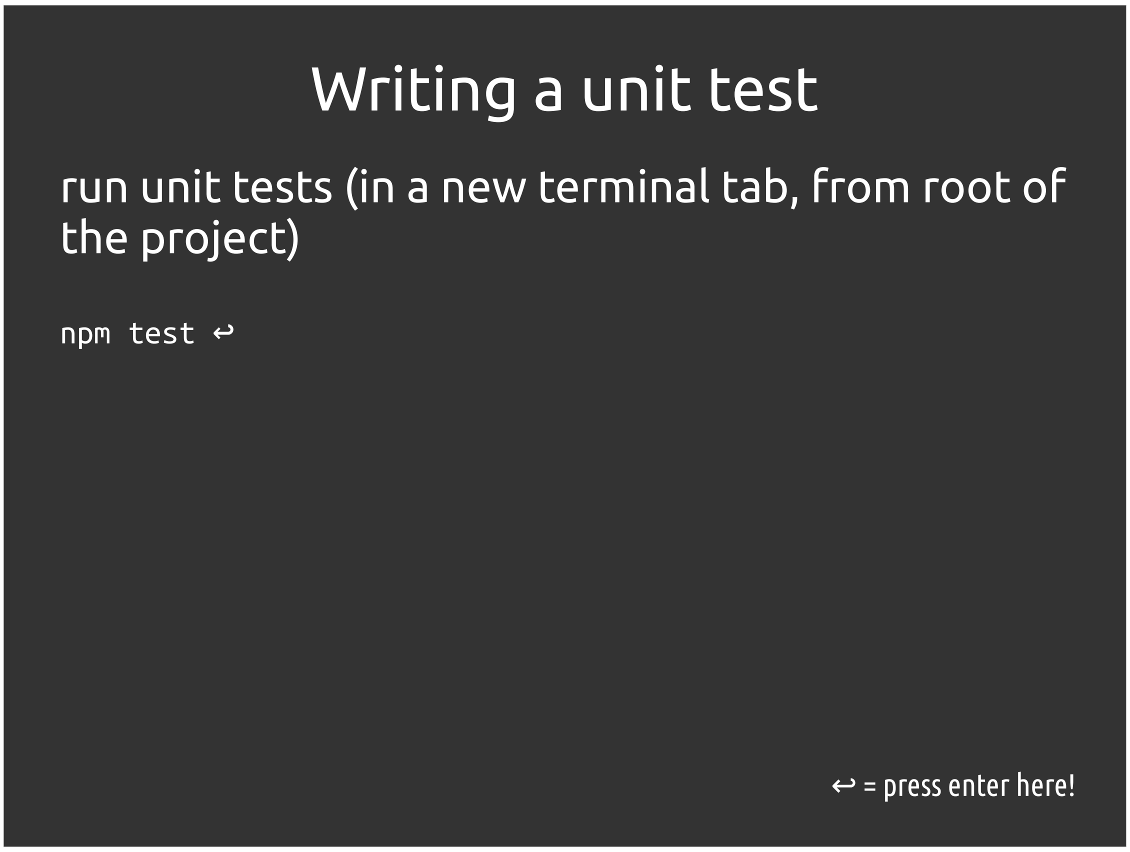
new file – components/articles/articles-services_test.js

```
'use strict';

describe('myApp.articles module', function() {
  beforeEach(module('myApp.articles', 'ngResource'));

  describe('articles service', function() {

    it('should make the Articles factory available',
      inject(function(Articles, $resource) {
        expect(Articles).toBeDefined();
    }));

  });
});
```

../karma.conf.js – add to files array

```
'app/bower_components/angular-resource/angular-resource.js',
```

- we write a human readable description of what the test is testing
- we write some expect statements, these are the actual tests that will suceed or fail



Writing a unit test

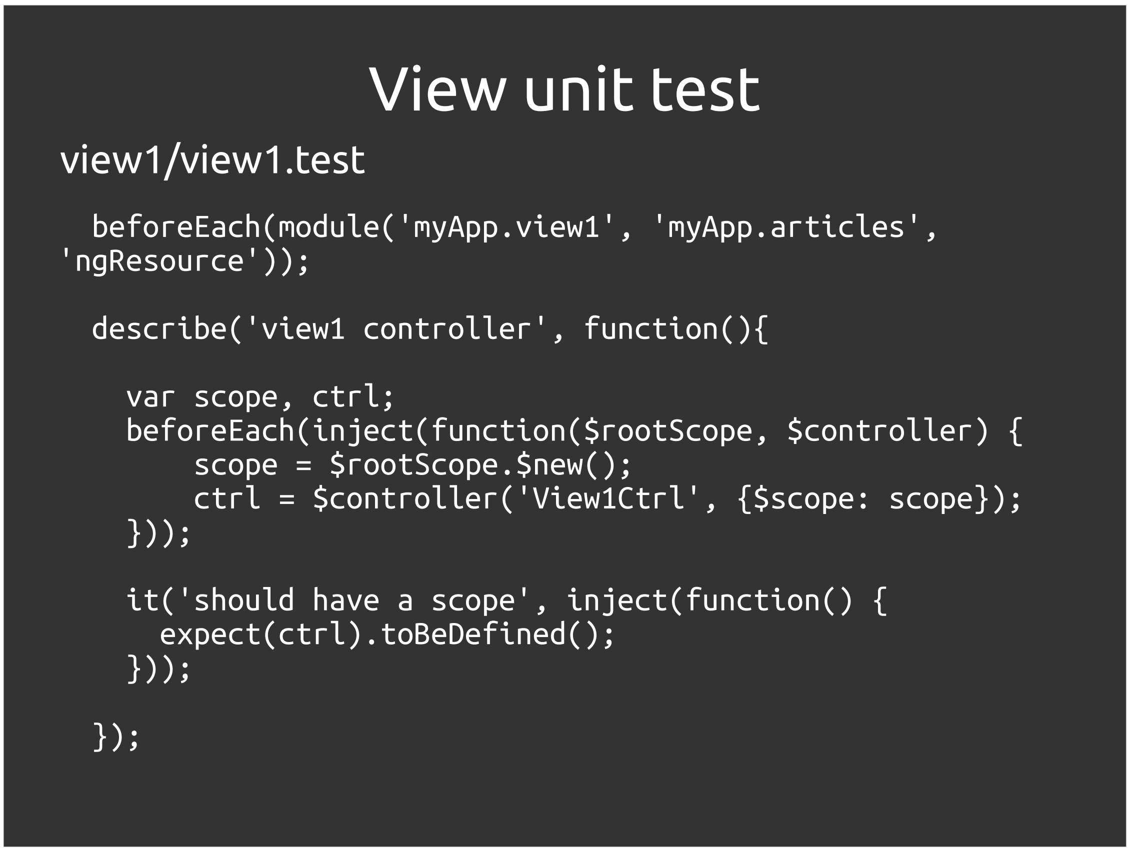
run unit tests (in a new terminal tab, from root of the project)

```
npm test ↩
```

↩ = press enter here!

- we run the tests in a command line environment.

The tests will rerun themselves automatically whenever we refresh the chrome instance the tests spawn

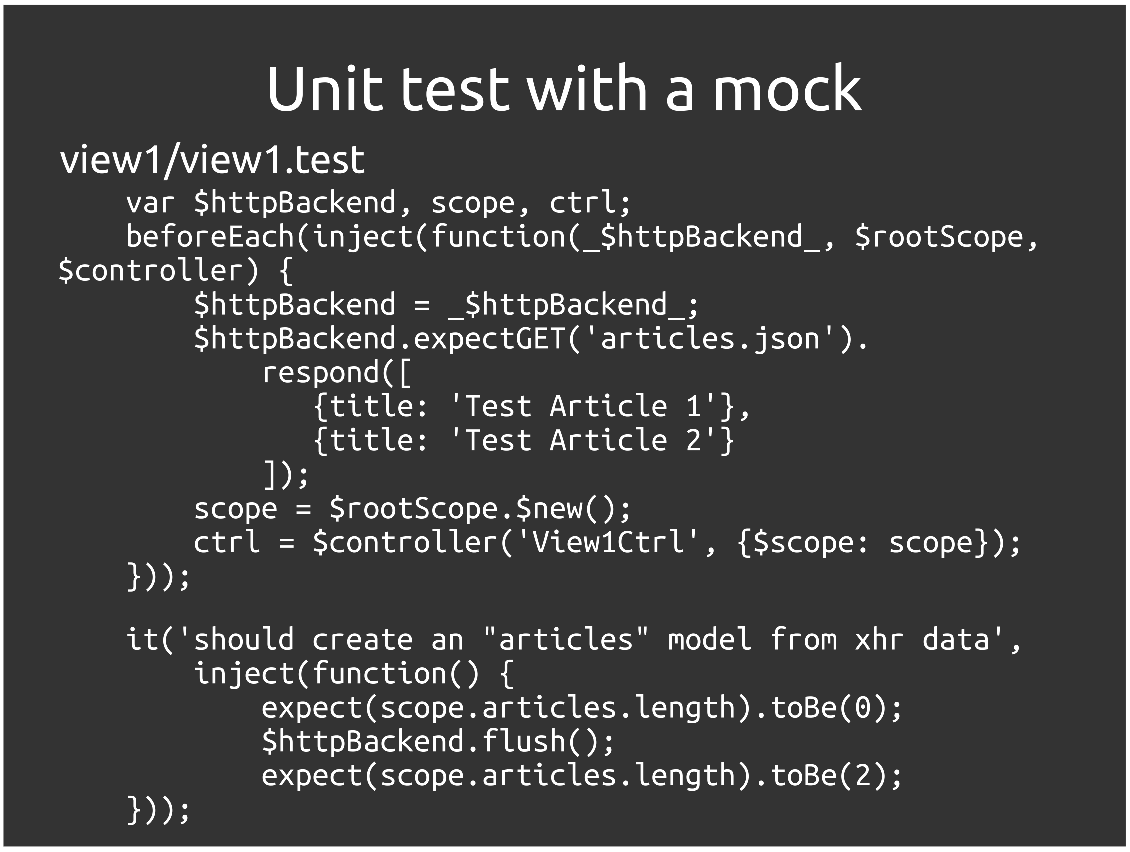


View unit test

view1/view1.test

```
  beforeEach(module('myApp.view1', 'myApp.articles',
'ngResource'));

  describe('view1 controller', function(){

    var scope, ctrl;
    beforeEach(inject(function($rootScope, $controller) {
        scope = $rootScope.$new();
        ctrl = $controller('View1Ctrl', {$scope: scope});
    }));

    it('should have a scope', inject(function() {
      expect(ctrl).toBeDefined();
    }));

  });
```

The other unit test is broken because of the scope object we started passing in when we wrote our original hello world code


Unit test with a mock

view1/view1.test

```
    var $httpBackend, scope, ctrl;
    beforeEach(inject(function(_$httpBackend_, $rootScope,
$controller) {
        $httpBackend = _$httpBackend_;
        $httpBackend.expectGET('articles.json').
            respond([
               {title: 'Test Article 1'},
               {title: 'Test Article 2'}
            ]);
        scope = $rootScope.$new();
        ctrl = $controller('View1Ctrl', {$scope: scope});
    }));

    it('should create an "articles" model from xhr data',
        inject(function() {
            expect(scope.articles.length).toBe(0);
            $httpBackend.flush();
            expect(scope.articles.length).toBe(2);
    }));
```

- inject the controllers and services modules
- then we set up all the mock data we will need, creating fake versions of the various services we are injecting
- we set up a mock data response. It's better to custom create this to contain the minimum of data you need to test, so it's easier to refactor later
- if there were multiple api calls that we had to mock, they would have to be performed in the correct order
- we flush the mock http service we created to mimic the api request being performed
- we add the scope into the controller we are testing

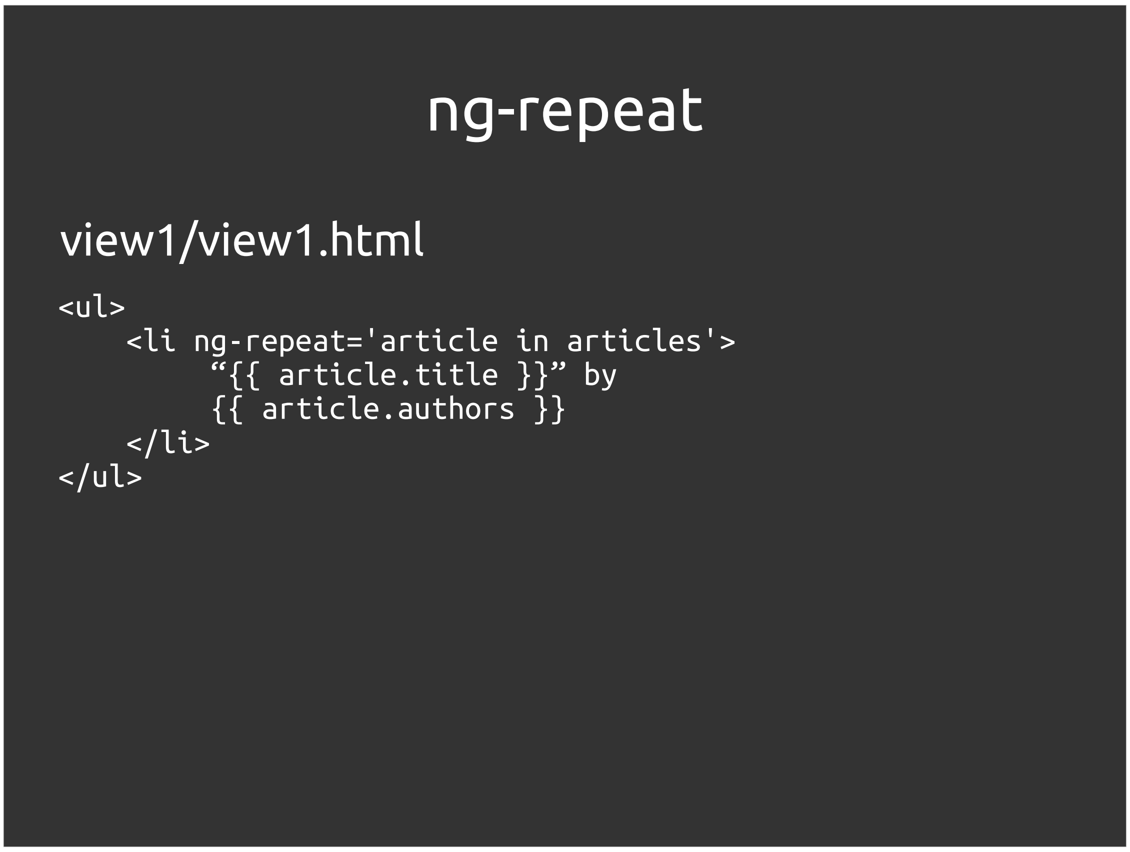
getting stuck in

Sidenote:
Use the docs!

http://docs.angularjs.org/api/

Built-in directives

- LOTS of built in cool things you can do
- better to show you than to tell you

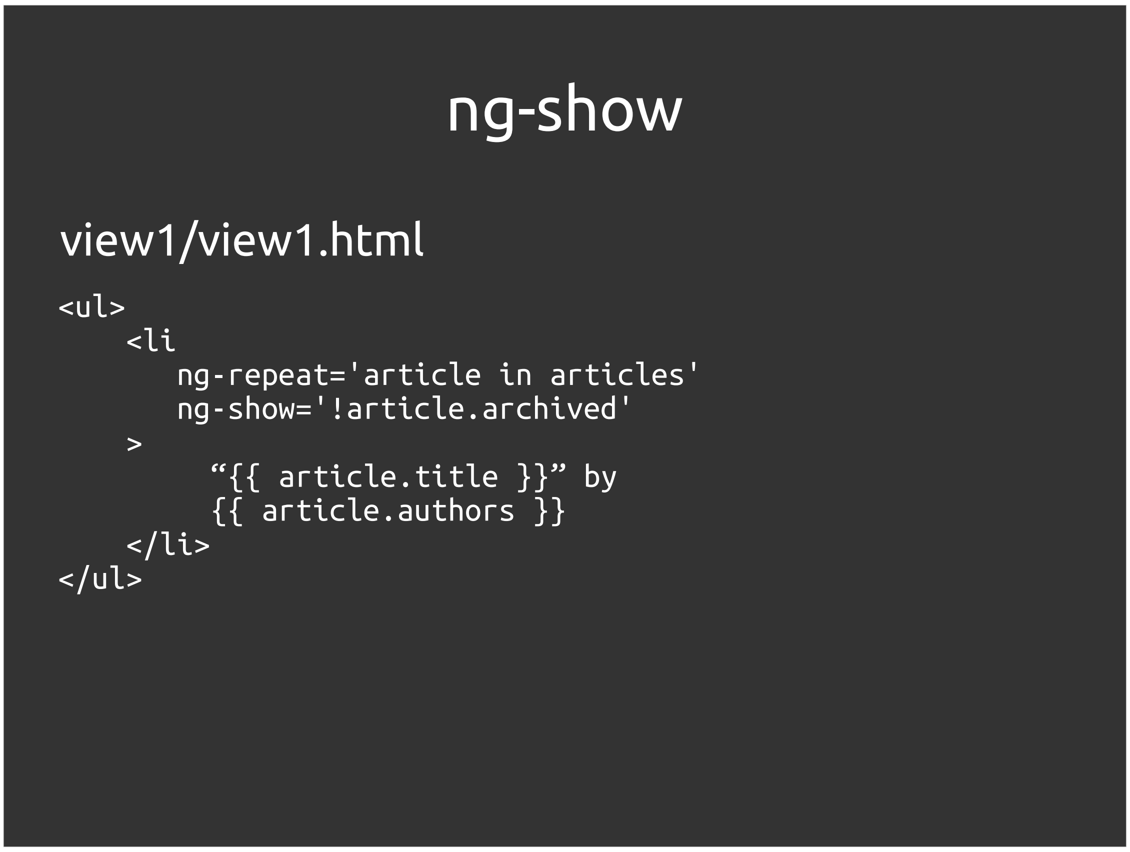


ng-repeat

view1/view1.html

```
<ul>
    <li ng-repeat='article in articles'>
         “{{ article.title }}” by
         {{ article.authors }}
    </li>
</ul>
```

- Essentially a for each loop
- Creates a child scope for 'article' that is descended from $scope

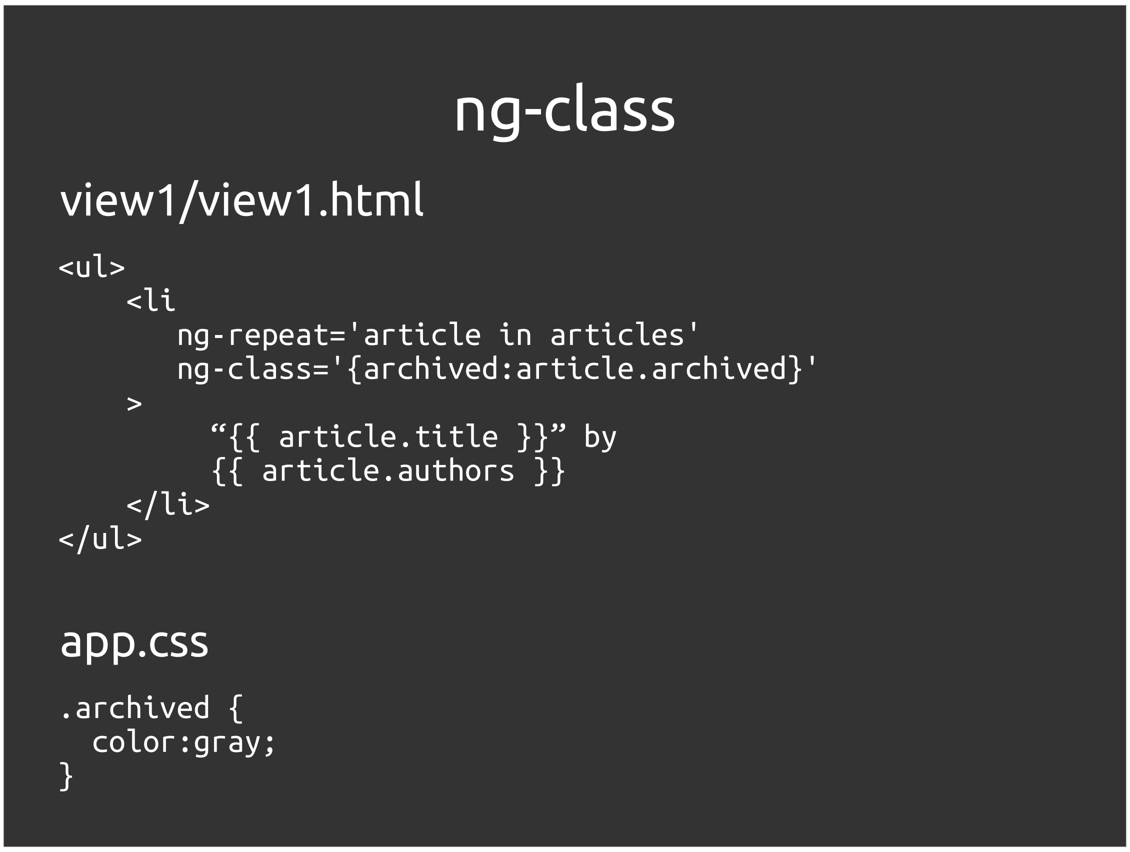


ng-show

view1/view1.html

```
<ul>
    <li
       ng-repeat='article in articles'
       ng-show='!article.archived'
    >
         “{{ article.title }}” by
         {{ article.authors }}
    </li>
</ul>
```

- ng-show adds an inline 'display:none' style to the element
- the attribute is evaluated as either true or false and displayed accordingly
- if you really need the element to not be there, use ng-if or ng-switch instead (but be aware these have more performance implications as removing elements from the DOM is costly)



ng-class

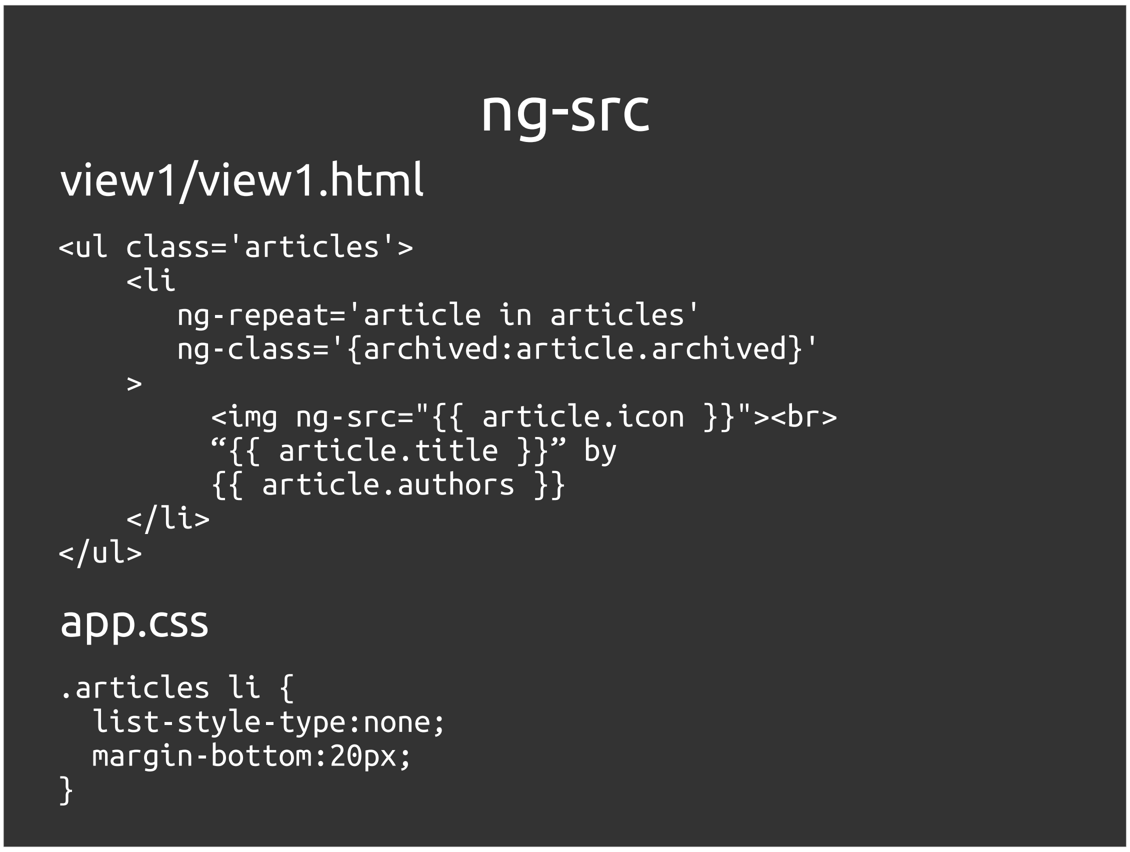
view1/view1.html

```
<ul>
    <li
       ng-repeat='article in articles'
       ng-class='{archived:article.archived}'
    >
         “{{ article.title }}” by
         {{ article.authors }}
    </li>
</ul>
```

app.css

```
.archived {
  color:gray;
}
```

- swap out the ng-show for an ng-class
- the key is the name of the class you want to get added, the value is the expression that will be evaluated as true or false to see if the class should be applied



ng-src

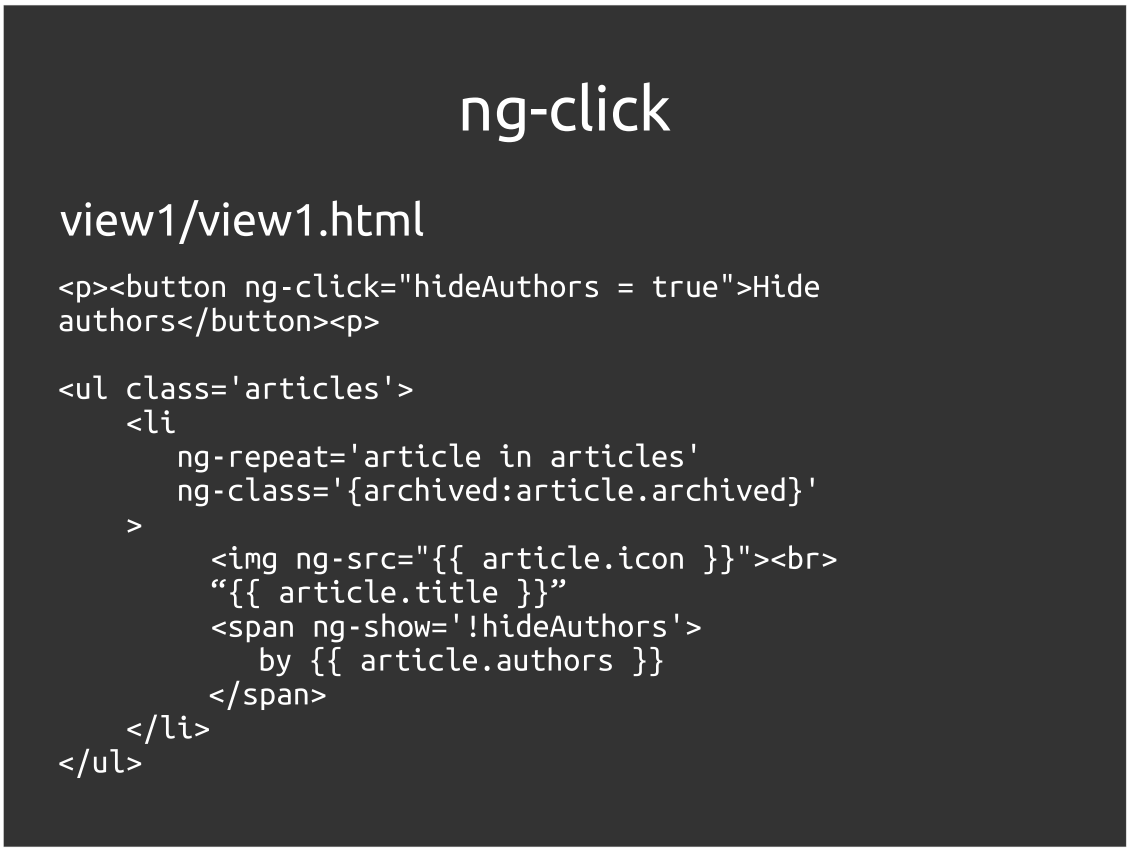
view1/view1.html

```
<ul class='articles'>
    <li
       ng-repeat='article in articles'
       ng-class='{archived:article.archived}'
    >
         <img ng-src="{{ article.icon }}"><br>
         “{{ article.title }}” by
         {{ article.authors }}
    </li>
</ul>
```

app.css

```
.articles li {
  list-style-type:none;
  margin-bottom:20px;
}
```

- rather than using a regular src attribute, where there are variables that must be evaluated we want to use ng-src instead – this prevents a broken HTTP request from occuring
- let's make the list at least a tiny bit less ugly while we're at it – add a class to the ul and a tiny bit more css

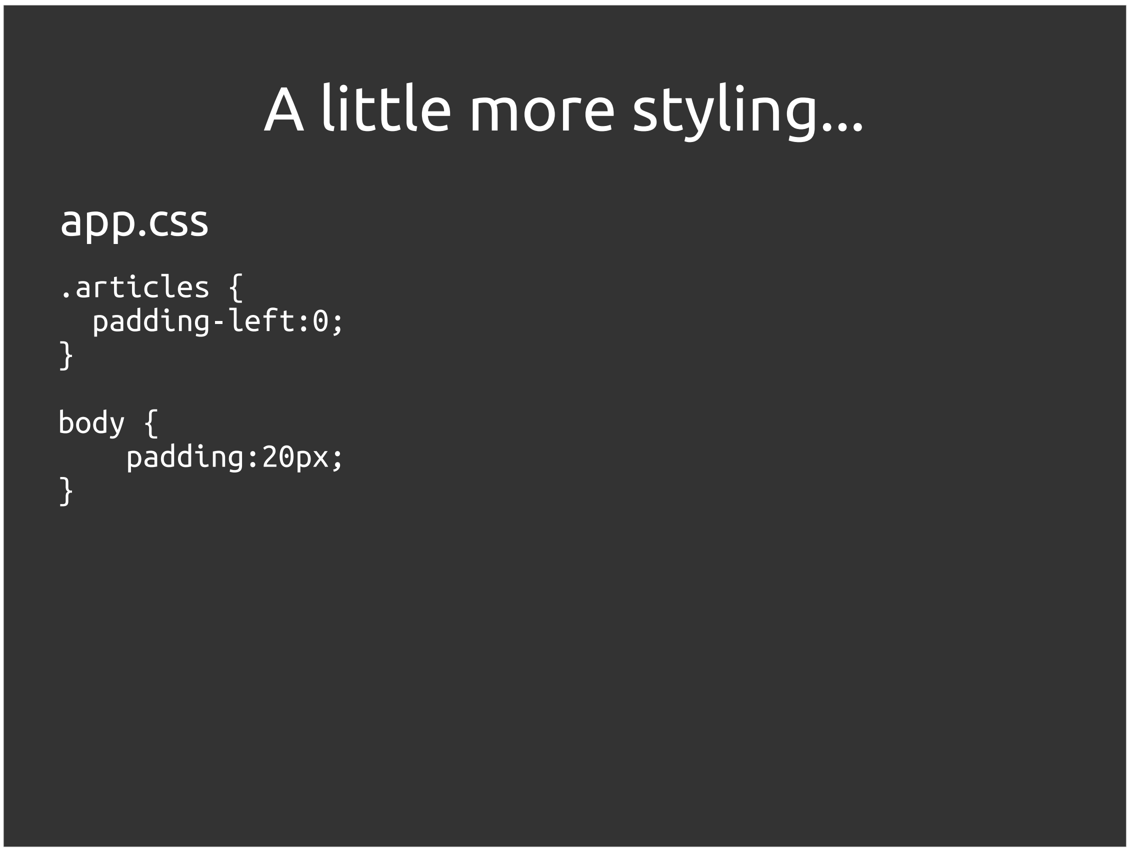


ng-click

view1/view1.html

```
<p><button ng-click="hideAuthors = true">Hide
authors</button><p>

<ul class='articles'>
    <li
       ng-repeat='article in articles'
       ng-class='{archived:article.archived}'
    >
         <img ng-src="{{ article.icon }}"><br>
         “{{ article.title }}”
         <span ng-show='!hideAuthors'>
            by {{ article.authors }}
         </span>
    </li>
</ul>
```

- we can use a ng-click to set the variable in scope the ng-show further down is evaluating
- it's better to do !hideAuthors as opposed to showAuthors because we want the authors to be shown initially on page load
- note this is only hiding the author info, right now we don't have a way of turning it back on again unless we reload the page



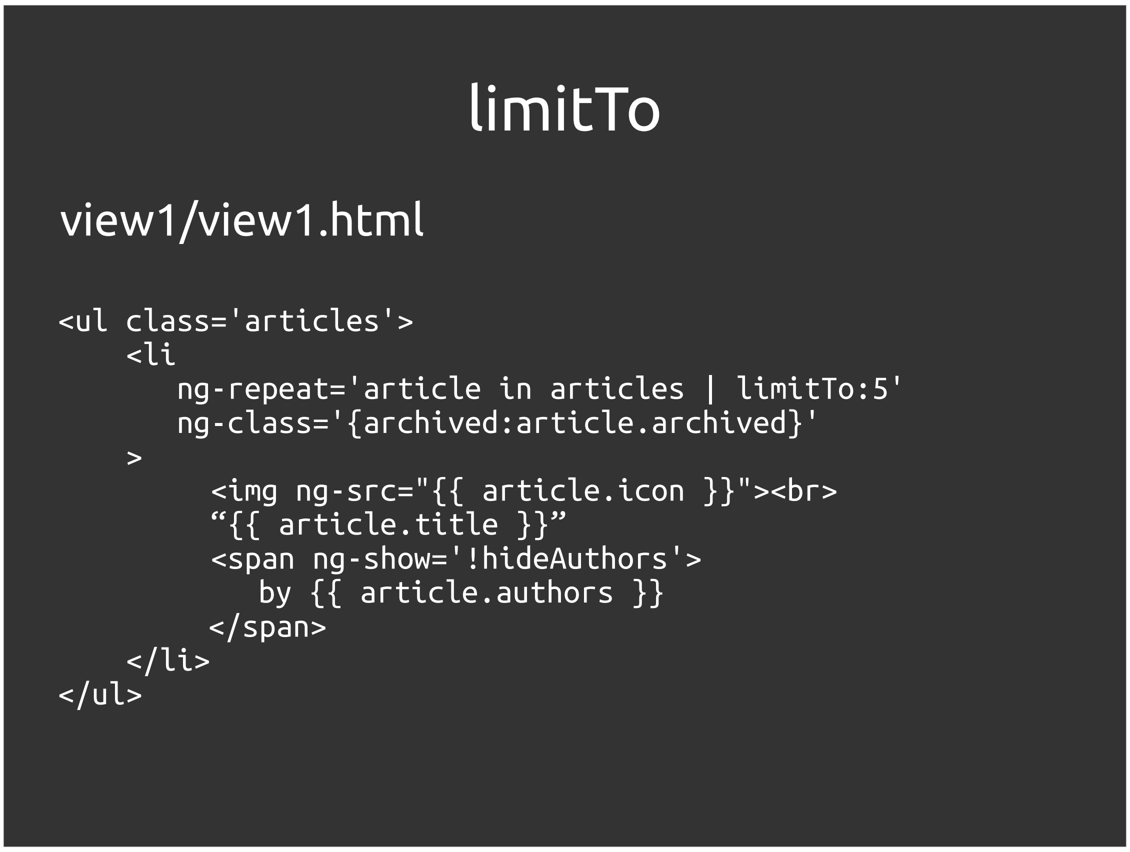
A little more styling...

app.css

```
.articles {
  padding-left:0;
}

body {
    padding:20px;
}
```

- we probably should have used a button element rather than a link

Built-in filters

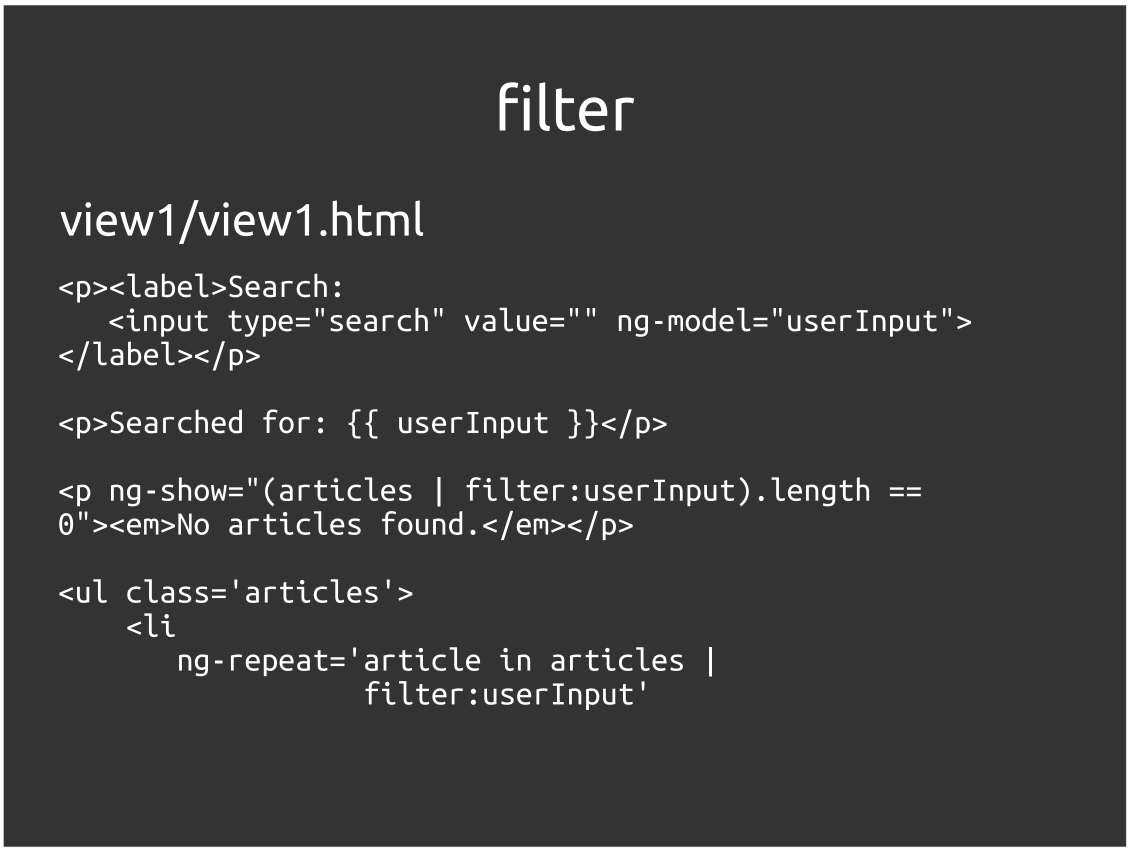


limitTo

view1/view1.html

```
<ul class='articles'>
    <li
       ng-repeat='article in articles | limitTo:5'
       ng-class='{archived:article.archived}'
    >
         <img ng-src="{{ article.icon }}"><br>
         “{{ article.title }}”
         <span ng-show='!hideAuthors'>
            by {{ article.authors }}
         </span>
    </li>
</ul>
```

- what if we only wanted to show a subset of articles? There are various filters we can chain onto the basic ng-repeat directive. Each filter is chained on using a pipe character
- Limit to 5 articles using the limitTo filter as shown

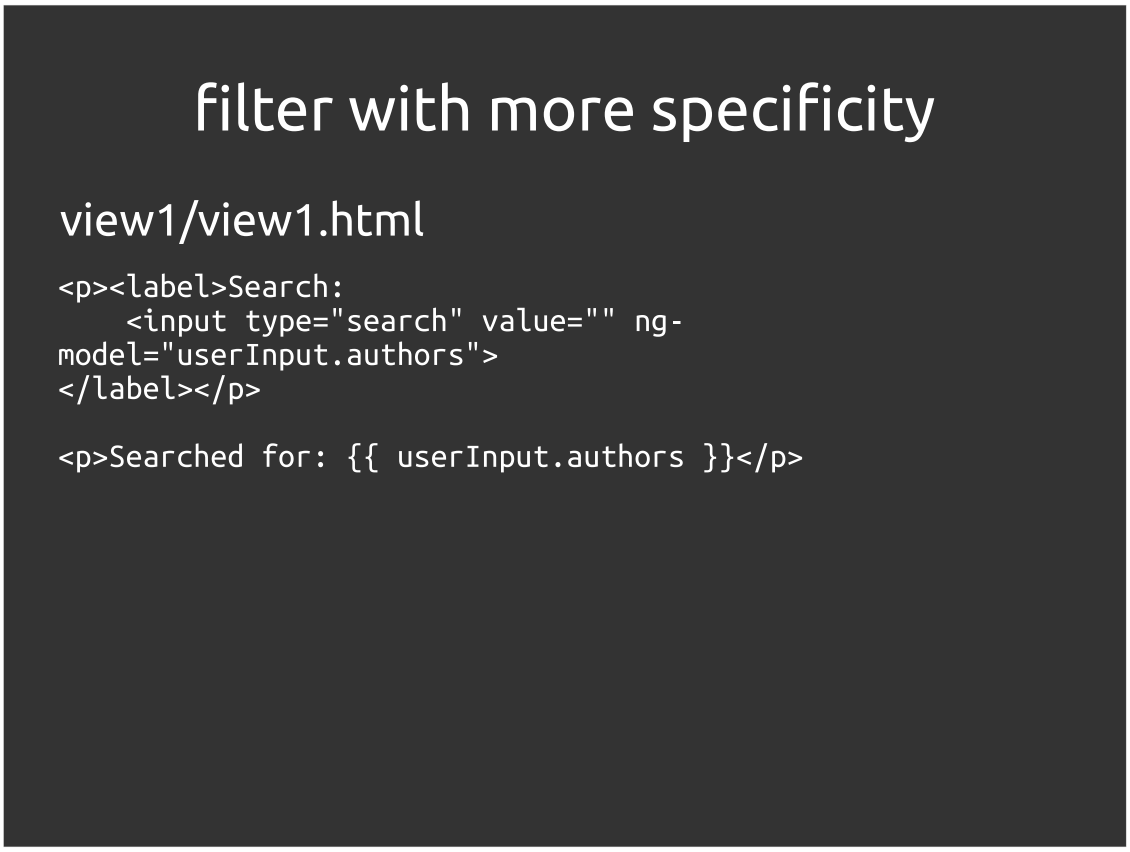


filter

view1/view1.html

```
<p><label>Search:
   <input type="search" value="" ng-model="userInput">
</label></p>

<p>Searched for: {{ userInput }}</p>

<p ng-show="(articles | filter:userInput).length ==
0"><em>No articles found.</em></p>

<ul class='articles'>
    <li
       ng-repeat='article in articles |
                  filter:userInput'
```

- we can reduce the article list based on user input to a text box that we set up as a ng-model
- as its value changes the two statements that do a filter based on userInput are updated automatically
- setting up two filters that do exactly the same thing like this is a bit wasteful – we could have created another scope variable instead to reuse
- filters can be used within controllers too by passing in the filter module

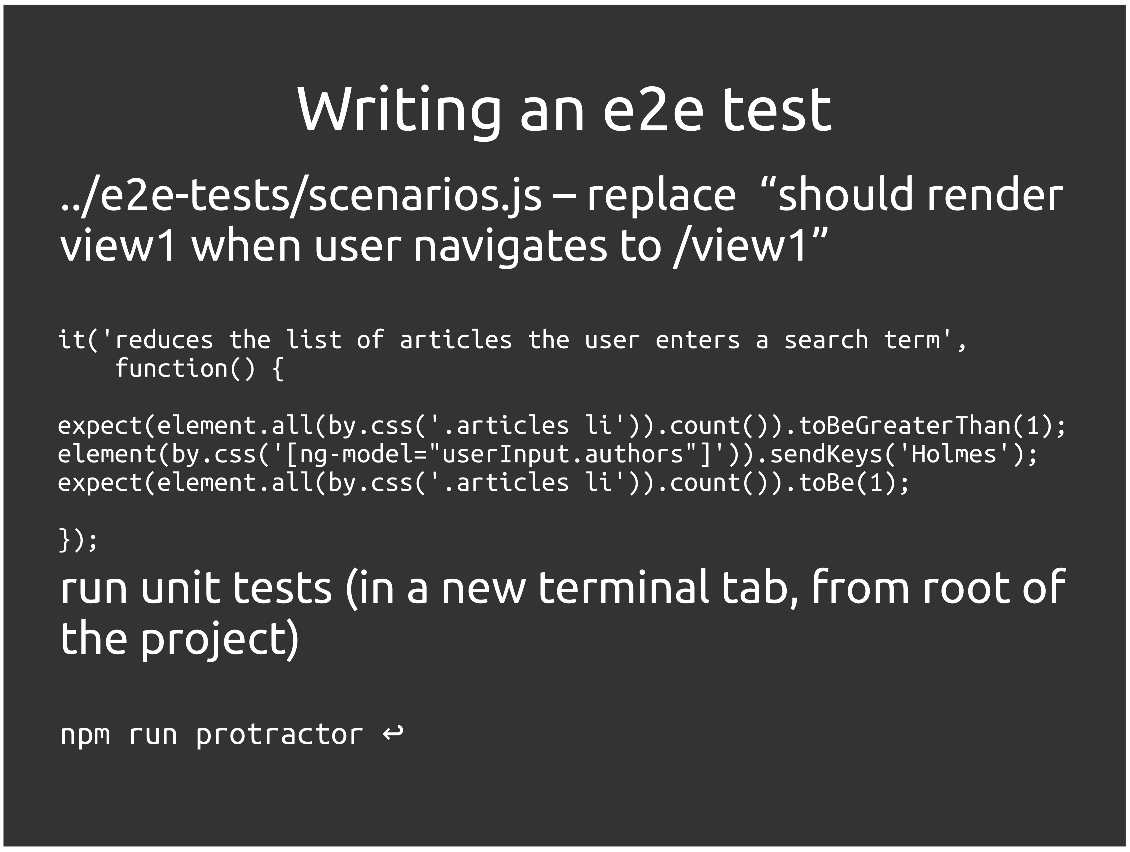


filter with more specificity

view1/view1.html

```
<p><label>Search:
    <input type="search" value="" ng-
model="userInput.authors">
</label></p>

<p>Searched for: {{ userInput.authors }}</p>
```

userInput is now an object with authors as an attribute.



Writing an e2e test

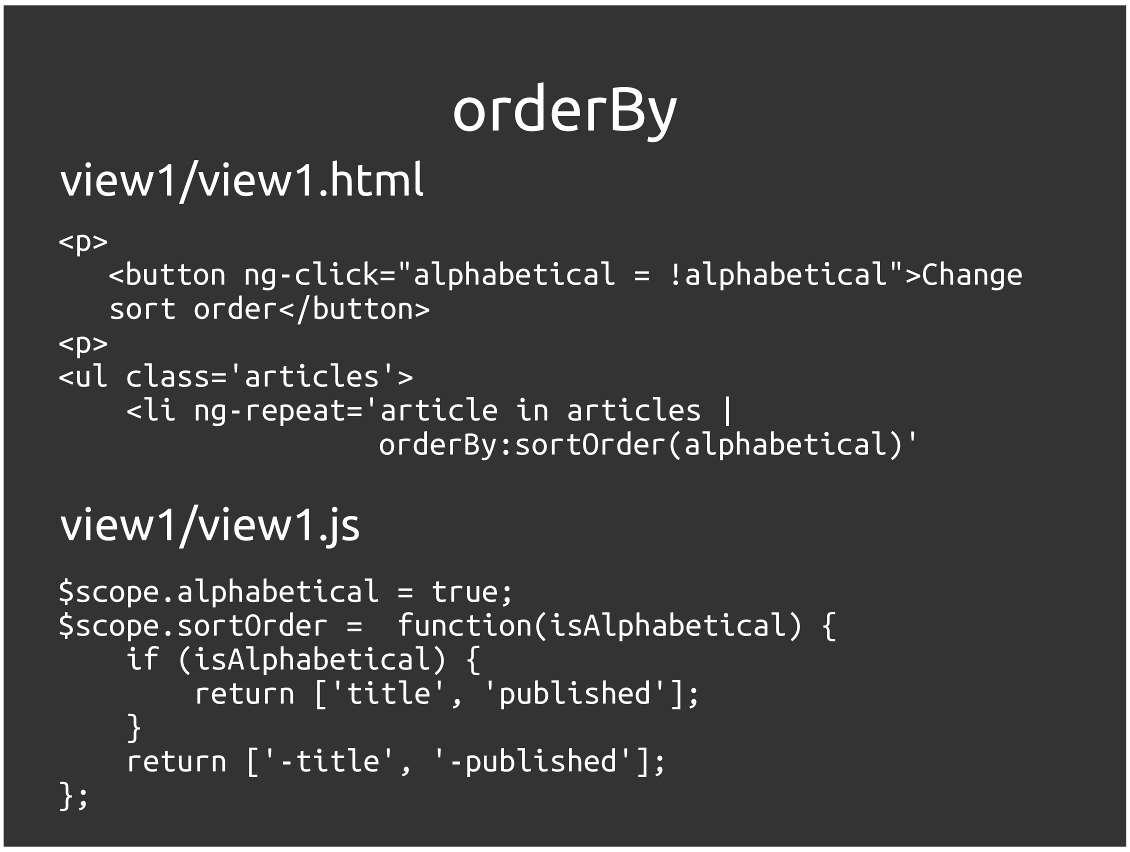
../e2e-tests/scenarios.js – replace "should render view1 when user navigates to /view1"

```
it('reduces the list of articles the user enters a search term',
    function() {

expect(element.all(by.css('.articles li')).count()).toBeGreaterThan(1);
element(by.css('[ng-model="userInput.authors"]')).sendKeys('Holmes');
expect(element.all(by.css('.articles li')).count()).toBe(1);

});
```

run unit tests (in a new terminal tab, from root of the project)

```
npm run protractor ↩
```

- find out more about Jasmine matchers at https://github.com/pivotal/jasmine/wiki/Matchers
-note that unlike the AngularJS tutorial, this uses Protractor https://github.com/angular/protracto, which uses WebDriverJS https://code.google.com/p/selenium/wiki/WebDriverJs , which is the JS bindings for Selenium



orderBy

view1/view1.html

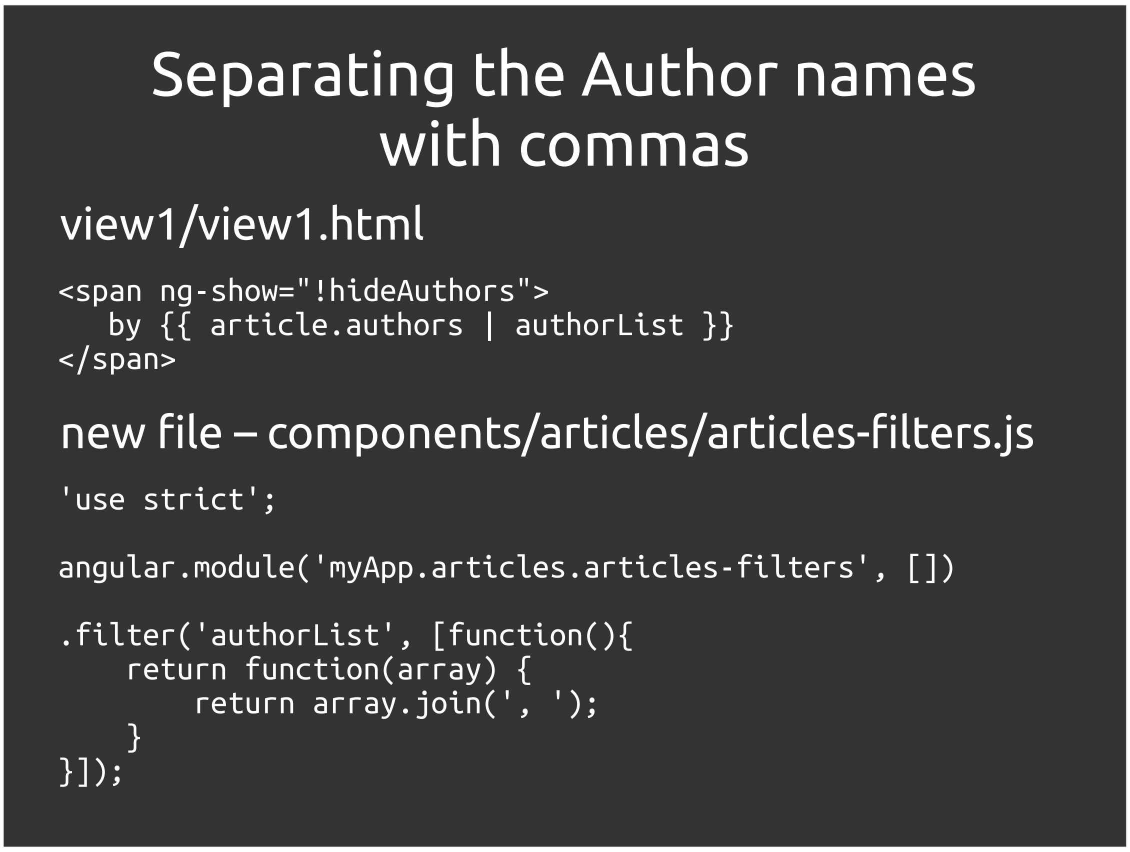
```
<p>
   <button ng-click="alphabetical = !alphabetical">Change
   sort order</button>
<p>
<ul class='articles'>
    <li ng-repeat='article in articles |
                   orderBy:sortOrder(alphabetical)'
```

view1/view1.js

```
$scope.alphabetical = true;
$scope.sortOrder =  function(isAlphabetical) {
    if (isAlphabetical) {
        return ['title', 'published'];
    }
    return ['-title', '-published'];
};
```

- Theres's a few things going on here
- We have defined the sortOrder as a function in the controller. We pass in the current value of $scope.alphabetical (this will get updated on the fly as the scope variable changes), and then depending on if it is true or false we return an array of attributes for the Article to use as the sort keys. The first item in the array is the primary sort key, and then if there are duplicate titles we would use the 'published' date as the secondary sort key
- Any sort keys that have a hyphen prepended are used to do the sort in reverse
- We also set up an ng-click link to change $scope.alphabetical to the opposite of what it is currently set as
- And in the controller we set up $scope.alphabetical to be true initially.

Writing our own filter



Separating the Author names with commas

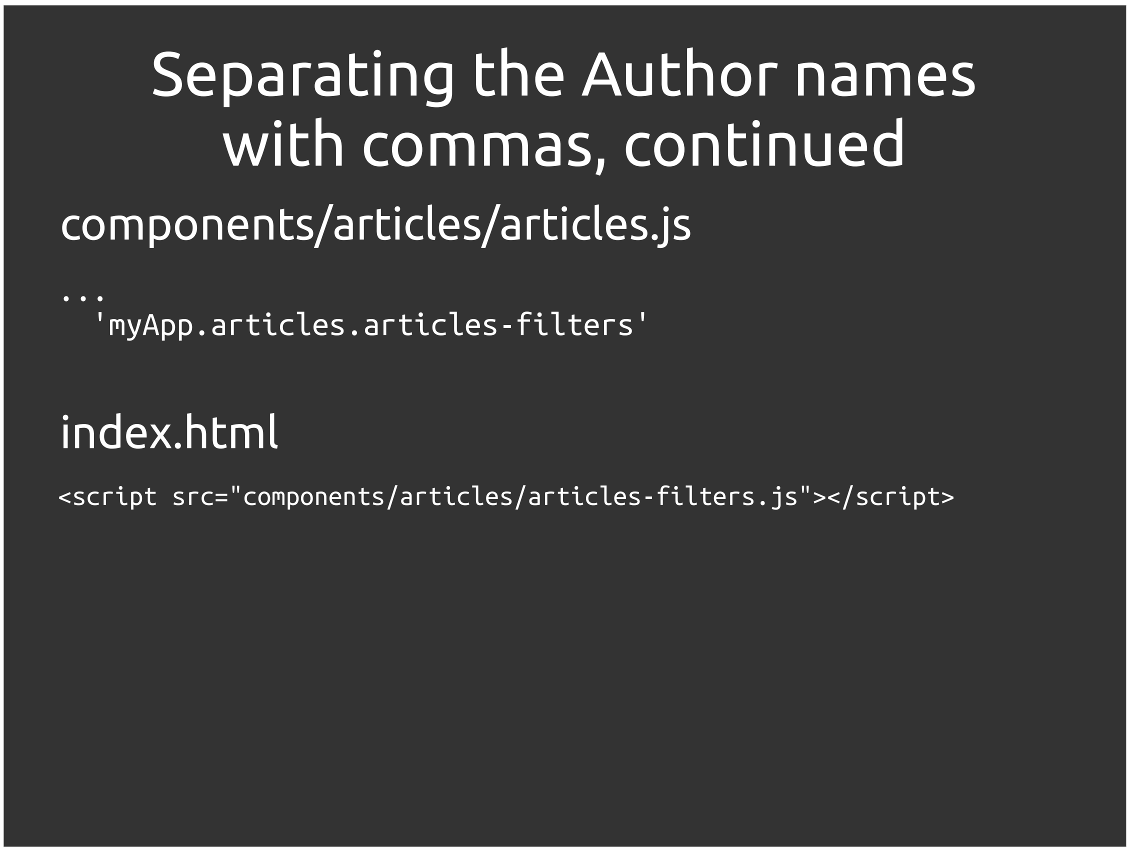
view1/view1.html

```
<span ng-show="!hideAuthors">
   by {{ article.authors | authorList }}
</span>
```

new file – components/articles/articles-filters.js

```
'use strict';

angular.module('myApp.articles.articles-filters', [])

.filter('authorList', [function(){
    return function(array) {
        return array.join(', ');
    }
}]);
```

- This is a pretty crude filter, we could make it more sophisticated by putting in some guard code to check an array is being passed in, or to add an 'and' for the final author etc



Separating the Author names with commas, continued

components/articles/articles.js

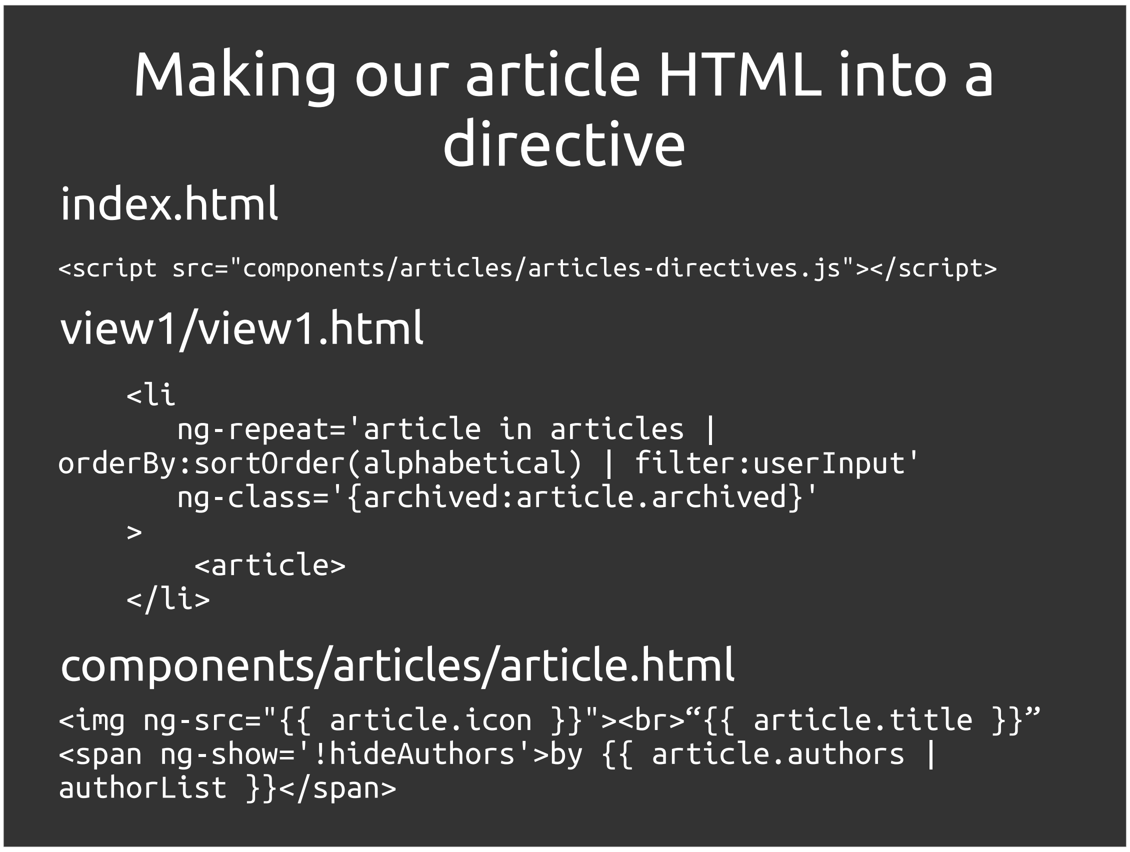
```
...
  'myApp.articles.articles-filters'
```

index.html

```
<script src="components/articles/articles-filters.js"></script>
```

- our articles manifest needs updating

Writing our own directive



Making our article HTML into a directive

index.html

```
<script src="components/articles/articles-directives.js"></script>
```

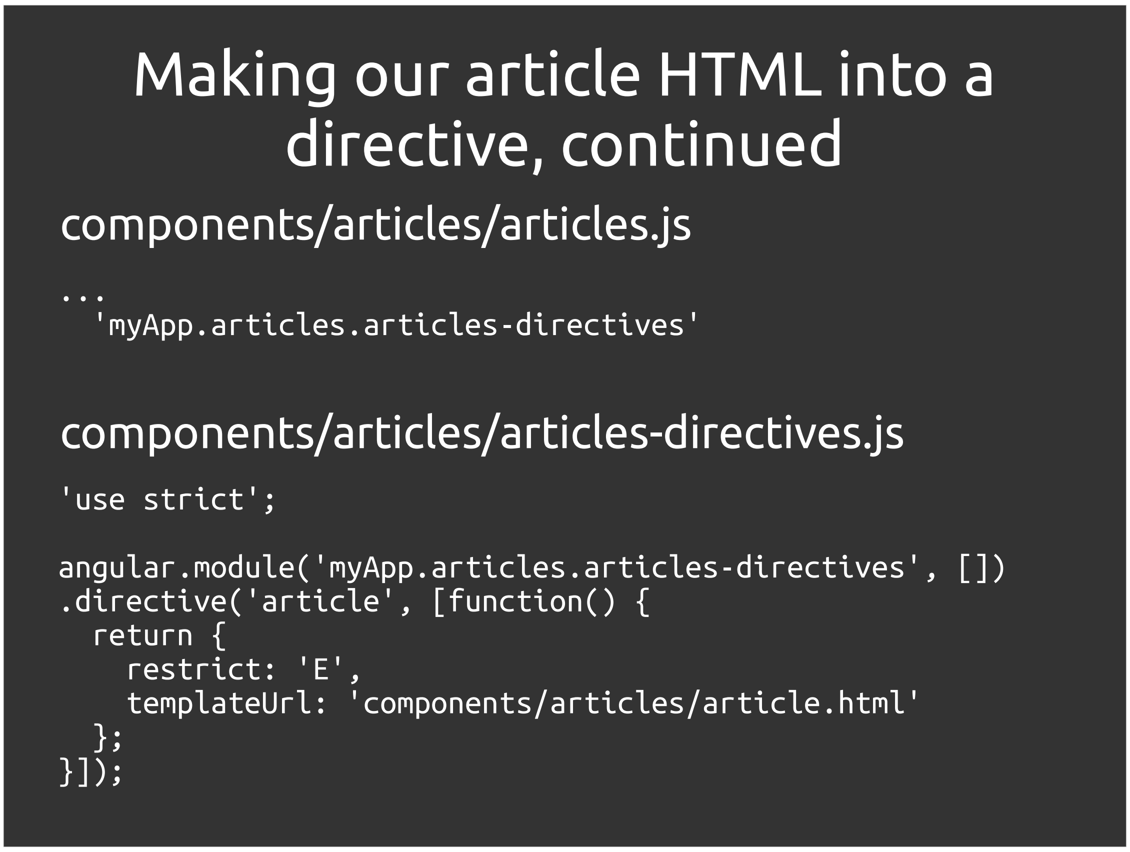
view1/view1.html

```
    <li
       ng-repeat='article in articles |
orderBy:sortOrder(alphabetical) | filter:userInput'
       ng-class='{archived:article.archived}'
    >
        <article>
    </li>
```

components/articles/article.html

```
<img ng-src="{{ article.icon }}"><br>“{{ article.title }}”
<span ng-show='!hideAuthors'>by {{ article.authors |
authorList }}</span>
```

- We'll need a new file for our directives theat we need to wire in to the index page
- We replace the template snippet within our ng-repeat li with our custom tag. There are a variety of ways we could structure this tag
- The HTML that used to be there now becomes its own template. We could achieve a similar effect to this rather naïve directive by just using a **partial template include**, but this is a good illustration of a simple directive



Making our article HTML into a directive, continued

components/articles/articles.js

```
...
  'myApp.articles.articles-directives'
```

components/articles/articles-directives.js

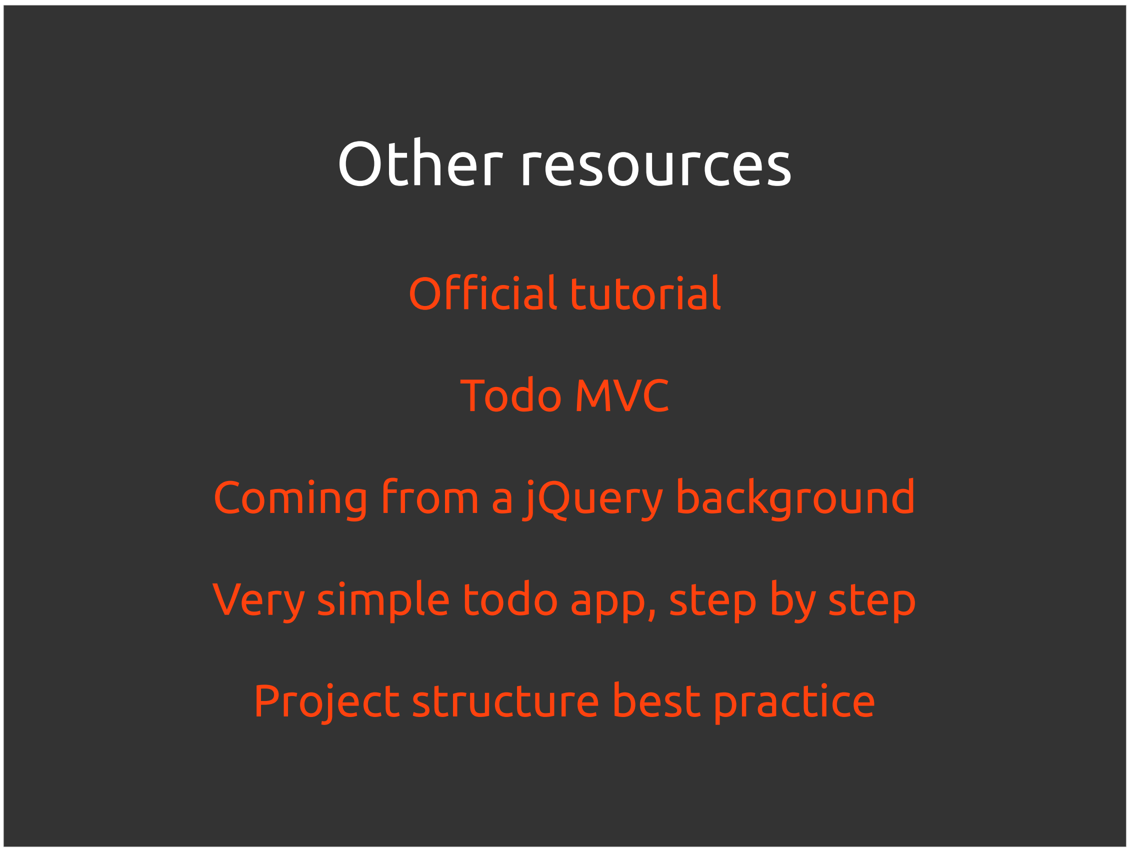
```
'use strict';

angular.module('myApp.articles.articles-directives', [])
.directive('article', [function() {
  return {
    restrict: 'E',
    templateUrl: 'components/articles/article.html'
  };
}]);
```

- our articles manifest needs updating
- our new file contains the directive. There are a few things going on here:
  - this directive is restricted to a particular way of being invoked, via E – elements. If we added A we could invoke it via attributes on regular elements
  - there is no scope specified so we inheirit the parent scope by default. If we wanted this directive to be reusable in lots of place we might consider making the scope isolated
  - we are referencing a template file. We could have just specified a string here rather than a URL. We can also transclude content, wrapping it in other markup. We also might write driectives that handle e.g. events and don't output any markup at all
  - We could have attached controllers to the directive so that it has functionality associated to it

Other built in stuff of interest

Services: location, window, q, animate, sanitize
Filters: date, currency, lowercase
Directives: paste, switch, angularUI

...loads more, read the docs!



Other resources

Official tutorial

Todo MVC

Coming from a jQuery background

Very simple todo app, step by step

Project structure best practice

http://docs.angularjs.org/tutorial/index

http://todomvc.com/architecture-examples/angularjs/

http://stackoverflow.com/questions/14994391/how-do-i
-think-in-angularjs-if-i-have-a-jquery-background

https://github.com/jenofdoom/summer-of-tech-js-boo
tcamp

https://docs.google.com/document/d/1XXMvReO8-
Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/mobilebasi
c?pli=1