ANGULARJS

by Google

Intro to SPA framework

Jussi Pohjolainen

Agenda

- Web Development and SPA
- Intro to AngularJS
- Recap of JS
- Getting Started with AngularJS – Directives, Filters and Data Binding
- View, Controllers and Scope
- Modules, Routes, Services
- Ajax and RESTful Web Services
- Animations and Testing

Rise of the Responsive Single Page App

Image: http://johnpolacek.github.io/scrolldeck.js/decks/responsive/

Responsive

- Unified across experiences
- Can be embedded as mobile app
- Better deployment and & maintanence
- Mobile users need to get access to everything

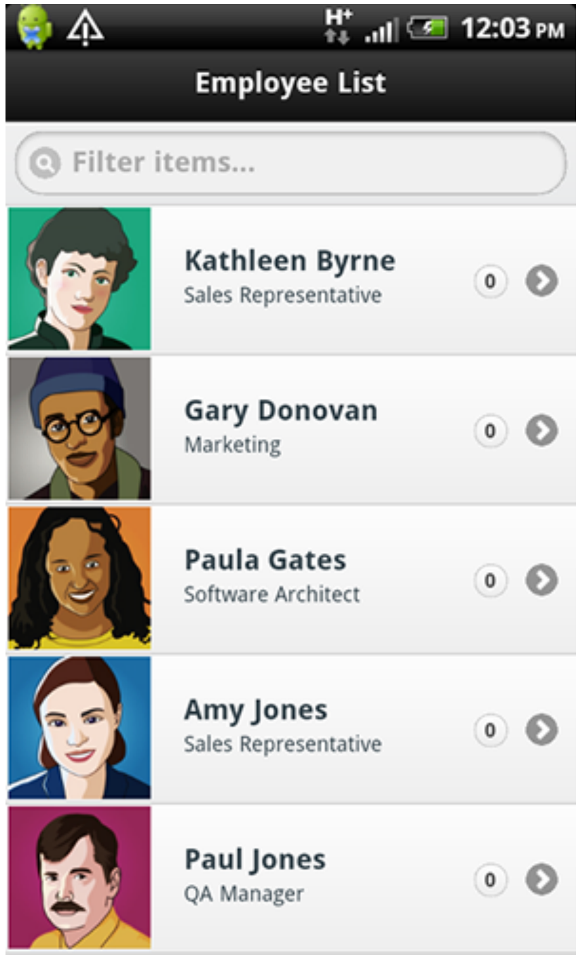


Image: http://coenraets.org/blog/wp-content/uploads/2011/10/directory11.png

Single-page Applications (SPA)

- Web app that fits on **a single web page**
  - Fluid UX, like desktop app
  - Examples like Gmail, Google maps
- Html page contains **mini-views** *(*HTML Fragments*)* that can be loaded in the background
- **No reloading** of the page, **better UX**
- Requires handling of **browser history, navigation and bookmarks**

JavaScript

- SPAs are implemented using **JavaScript** and **HTML**
- ECMAScript is a scripting language, standardized by Ecma International
- In Browsers, **ECMAScript is commonly called JavaScript**
  - **JavaScript = Native (EcmaScript) + Host objects (browser)**

Challenges in SPA

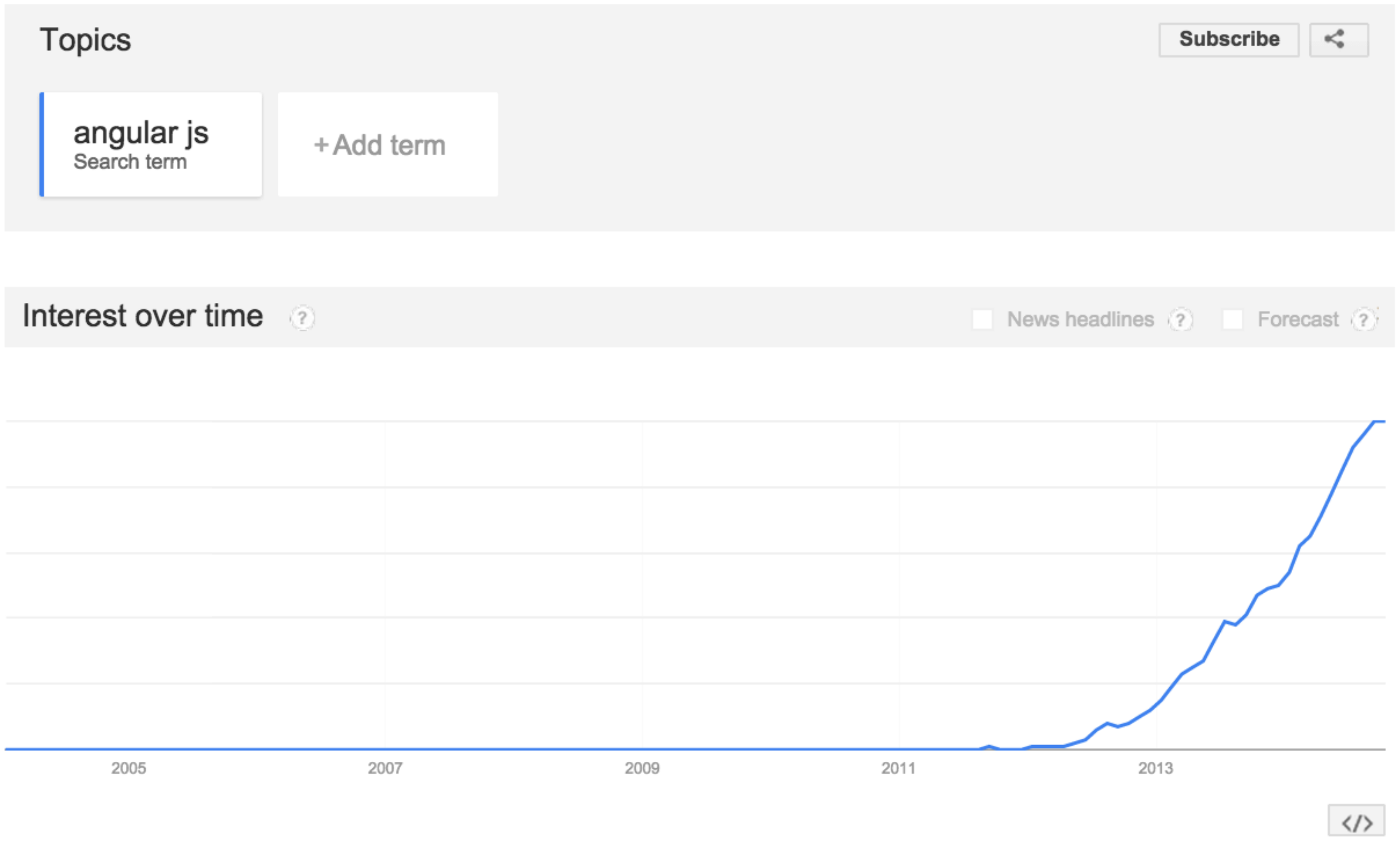
- **DOM Manipulation**
  - How to manipulate the view efficiently?
- **History**
  - What happens when pressing back button?
- **Routing**
  - Readable URLs?
- **Data Binding**
  - How bind data from model to view?
- **View Loading**
  - How to load the view?
- Lot of coding! You could **use a framework instead ...**

ANGULAR_JS

Angular JS

- **Single Page App Framework** for JavaScript
- Implements client-side **MVC** pattern
  - Separation of presentation from business logic and presentation state
- **No direct DOM** manipulation, less code
- Support for all major browsers
- Supported by Google
- Large and fast growing community

Interest in AngularJS



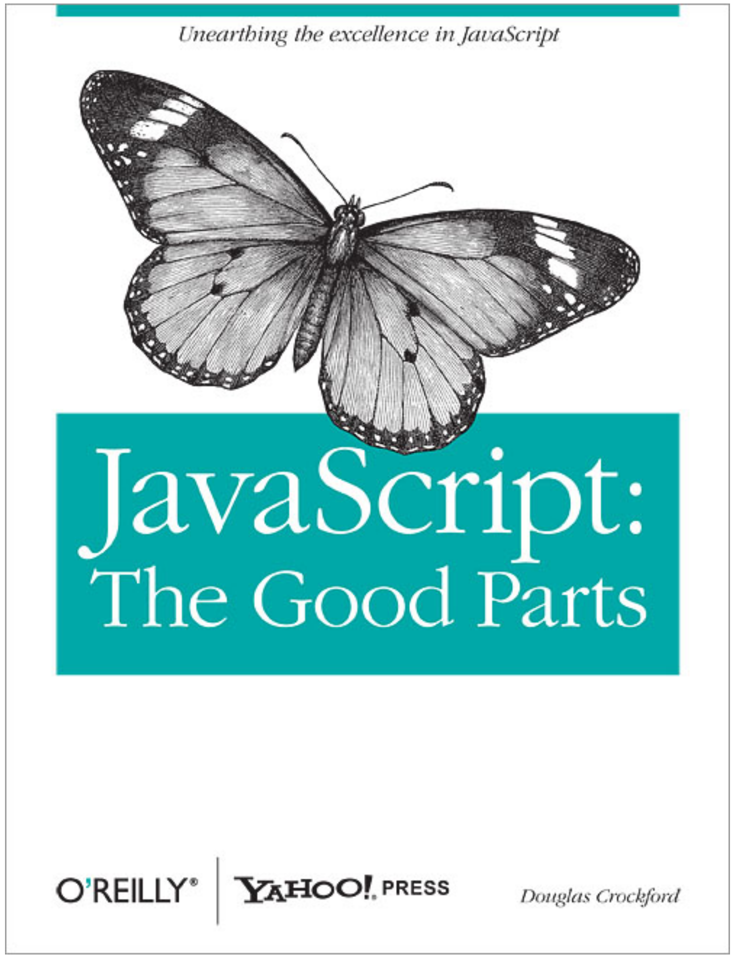
AngularJS – Main Concepts

- Templates
- Directives
- Expressions
- Data binding
- Scope
- Controllers
- Modules
- Filters
- Services
- Routing

RECAP OF JAVASCRIPT

Recommended Reading

- Recommended reading
  - **JavaScript: The Good Parts** by Douglas Crockford
  - **JavaScript: The Definite Guide** by David Flanagan
  - **JavaScript Patterns:** Stoyan Stefanov


Unearthing the excellence in JavaScript
JavaScript: The Good Parts
O'REILLY®
YAHOO! PRESS
Douglas Crockford

Objects

- Everything (except basic types) **are objects**
  - Including **functions** and **arrays**
- Object contains **properties** and **methods**
  - Collection of name-value pairs
  - Names are strings, values can be anything
  - Properties and methods **can be added at runtime**
- Objects can inherit other objects

Object Literal

```
var customer = {name: "Jack", gender: "male"};

var circle1 = {radius: 9, getArea: someFunction};

var circle2 = {
   radius: 9,
   getRadius: function() {
        return this.radius;
   }
}
```

Properties at Runtime

- One of the simplest way to create object:

```
var obj = new Object();
obj.x = 10;
obj.y = 12;
obj.method = function() { … }
```

- This adds **at runtime** two properties to the obj – object!
- Object is built – in data type

“Class”

- You can create **constructor**-function in JavaScript

```
function Point() {
  this.x = 1;
  this.y = 1;
}
 var p = new Point();
```

Example

```
function Circle(radius)
{
        this.radius = radius;
        this.getArea = function()
        {
            return (this.radius * this.radius) * Math.PI;
        };
}

var myobj = new Circle(5);
document.write( myobj.getArea() );
```

Functions

- Every function in JS is **Function object**
  - Can be passed as arguments
  - Can store name / value pairs
  - Can be anonymous or named
- Usage (Don't use this, it's not efficient)

```
var myfunction = new Function("a","b", "return a+b;");
print( myfunction(3,3) );
```

Anonymous functions

- Functions can take functions as arguments
  - `fetchUrl( onSuccess, onError )`
- You can use **anonymous** functions
  - `fetchUrl( function() { ... }, function() { ... } );`

GETTING STARTED WITH ANGULAR_JS

First Example – Template

Template

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Title
    </title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="angular.min.js"></script>
  </head>
  <body>
    <!-- initialize the app -->
    <div ng-app>
      <!-- store the value of input field into a variable name -->
      <p>Name: <input type="text" ng-model="name"></p>
      <!-- display the variable name inside (innerHTML) of p -->
      <p ng-bind="name"></p>
    </div>
  </body>
</html>
```

Download this file from: https://angularjs.org/

Directive

Directive

Basic Concepts

- **1) Templates**
  - HTML with additional markup, directives, expressions, filters ...
- **2) Directives**
  - Extend HTML using `ng-app`, `ng-bind`, `ng-model`
- **3) Filters**
  - Filter the output: `filter`, `orderBy`, `uppercase`
- **4) Data Binding**
  - Bind model to view using expressions `{{ }}`

2) Directives

- **Directives** apply special behavior to attributes or elements in HTML
  - Attach behaviour, transform the DOM
- Some directives
  - `ng-app`
    - Initializes the app
  - `ng-model`
    - Stores/updates the value of the input field into a variable
  - `ng-bind`
    - Replace the text content of the specified HTML with the value of given expression

About Naming

- AngularJS HTML Compiler supports multiple formats
  - `ng-bind`
    - Recommended Format
  - `data-ng-bind`
    - Recommended Format to support HTML validation
  - `ng_bind, ng:bind, x-ng-bind`
    - Legacy, don't use

Lot of Built in Directives

- `ngApp`
- `ngClick`
- `ngController`
- `ngModel`
- `ngRepeat`
- `ngSubmit`
- `ngDblClick`
- `ngMouseEnter`
- `ngMouseMove`
- `ngMouseLeave`
- `ngKeyDown`
- `ngForm`

2) Expressions

- Angular **expressions** are JavaScript-like code snippets that are usually placed in bindings
  - `{{ expression }}.`
- Valid Expressions
  - `{{ 1 + 2 }}`
  - `{{ a + b }}`
  - `{{ items[index] }}`
- Control flow (loops, if) are not supported!
- You can use **filters** to format or filter data

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
  </head>
  <body>
    <div ng-app>
      <p>Number 1: <input type="number" ng-model="number1"></p>
      <p>Number 2: <input type="number" ng-model="number2"></p>
      <!-- expression -->
      <p>{{ number1 + number2 }}</p>
    </div>
  </body>
</html>
```

Directive

Directive

Expression

Valid HTML5 version

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
  </head>
  <body>
    <div data-ng-app="">
      <p>Number 1: <input type="number" data-ng-model="number1"></p>
      <p>Number 2: <input type="number" data-ng-model="number2"></p>
      <!-- expression -->
      <p>{{ number1 + number2 }}</p>
    </div>
  </body>
</html>
```

Add data-

Add data-

ng-init and **ng-repeat** directives

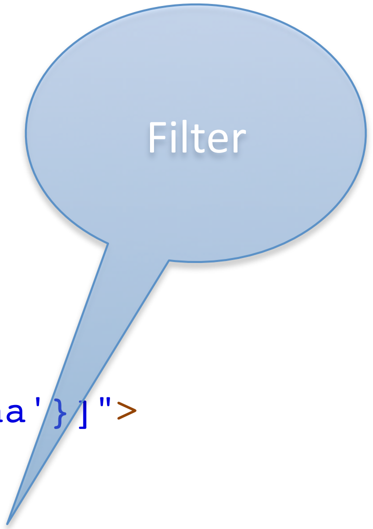
```
<html data-ng-app="">
<head>
  <title>Title</title>
  <meta charset="UTF-8" />
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-init="names = ['Jack', 'John', 'Tina']">
    <h1>Cool loop!</h1>
    <ul>
      <li data-ng-repeat="name in names">{{ name }}</li>
    </ul>
  </div>
</body>
</html>
```

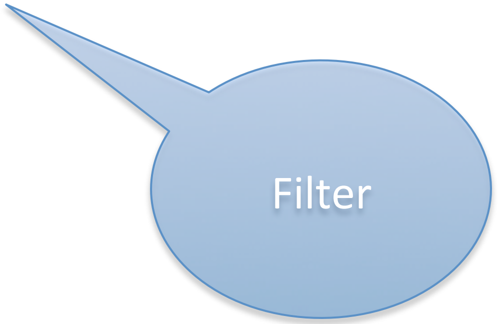
3) Filter

- With **filter,** you can **format or filter** the output
- **Formatting**
  - `currency, number, date, lowercase, uppercase`
- **Filtering**
  - `filter, limitTo`
- **Other**
  - `orderBy, json`

Using Filters - Example

```
<!DOCTYPE html>
<html data-ng-app="">
<head>

  <title>Title</title>
  <meta charset="UTF-8">
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-init="customers = [{name:'jack'}, {name:'tina'}]">
    <h1>Cool loop!</h1>
    <ul>
      <li data-ng-repeat="customer in customers | orderBy:'name'">
          {{ customer.name | uppercase }}</li>
    </ul>
  </div>
</body>
</html>
```





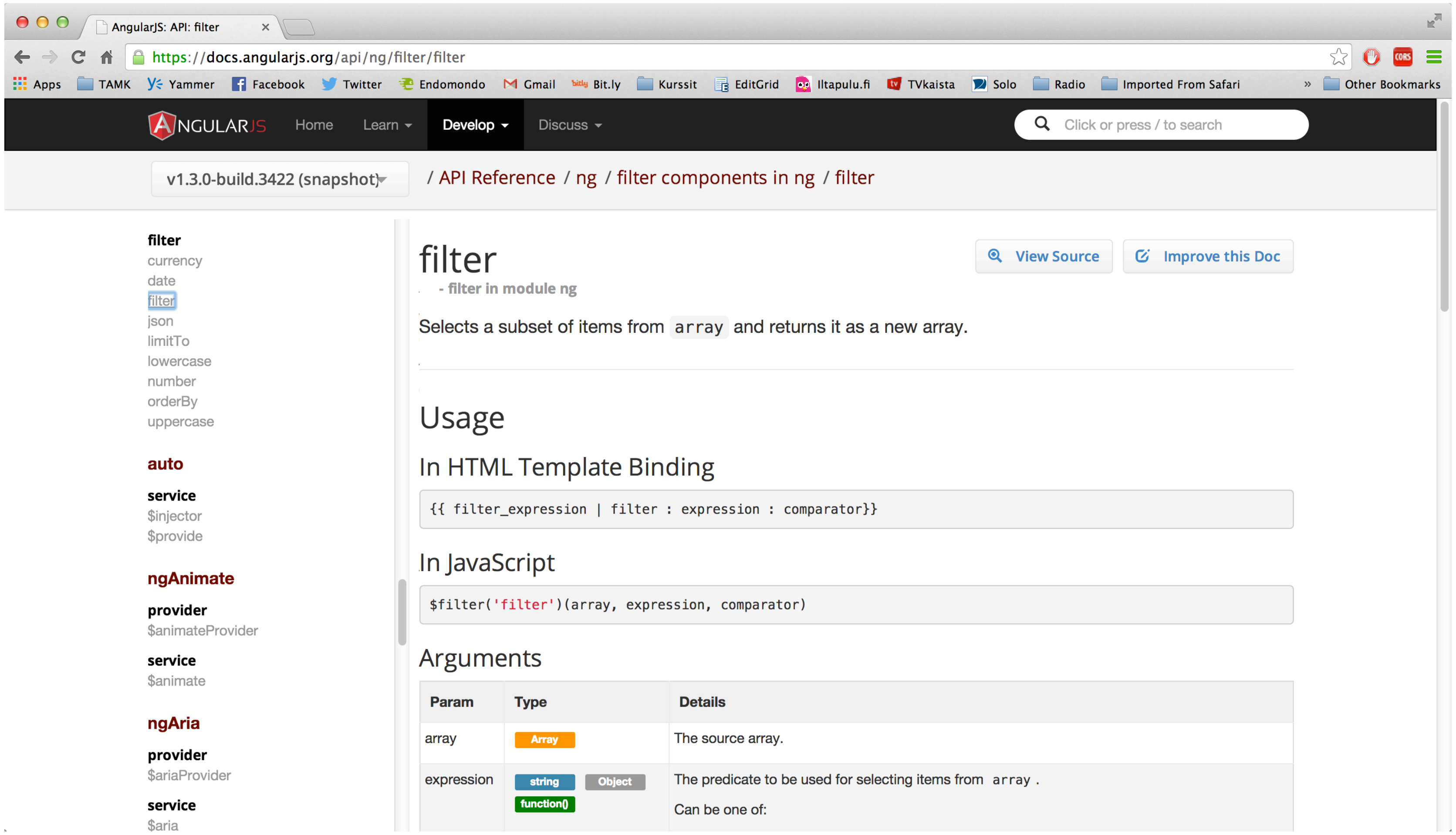
Using Filters - Example

```
<!DOCTYPE html>

<html data-ng-app="">
<head>
  <title>Title</title>
  <meta charset="UTF-8">
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>
<body>
  <div data-ng-init=
  "customers = [{name:'jack'}, {name:'tina'}, {name:'john'}, {name:'donald'}]">
    <h1>Customers</h1>

    <ul>
      <li data-ng-repeat="customer in customers | orderBy:'name' | filter:'john'">{{
      customer.name | uppercase }}</li>
    </ul>
  </div>
</body>
</html>
```

Using Filters – User Input Filters the Data

```
<!DOCTYPE html>

<html data-ng-app="">
<head>
  <title>Title</title>
  <meta charset="UTF-8">
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>
<body>
  <div data-ng-init=
  "customers = [{name:'jack'}, {name:'tina'}, {name:'john'}, {name:'donald'}]">
    <h1>Customers</h1>

    <input type="text" data-ng-model="userInput" />
    <ul>
      <li data-ng-repeat="customer in customers | orderBy:'name' | filter:userInput">{{
      customer.name | uppercase }}</li>
    </ul>
  </div>
</body>
</html>
```

API Reference



AngularJS: API: filter
https://docs.angularjs.org/api/ng/filter/filter
Apps TAMK Yammer Facebook Twitter Endomondo Gmail Bit.ly Kurssit EditGrid Iltapulu.fi TVkaista Solo Radio Imported From Safari Other Bookmarks
ANGULARJS Home Learn Develop Discuss
Click or press / to search
v1.3.0-build.3422 (snapshot)
/ API Reference / ng / filter components in ng / filter
View Source
Improve this Doc

filter
currency
date
filter
json
limitTo
lowercase
number
orderBy
uppercase

auto

service
$injector
$provide

ngAnimate

provider
$animateProvider

service
$animate

ngAria

provider
$ariaProvider

service
$aria

filter

- filter in module ng

Selects a subset of items from `array` and returns it as a new array.

Usage

In HTML Template Binding

```
{{ filter_expression | filter : expression : comparator}}
```

In JavaScript

```
$filter('filter')(array, expression, comparator)
```

Arguments

Param	Type	Details
array	Array	The source array.
expression	string Object function()	The predicate to be used for selecting items from `array` . Can be one of:

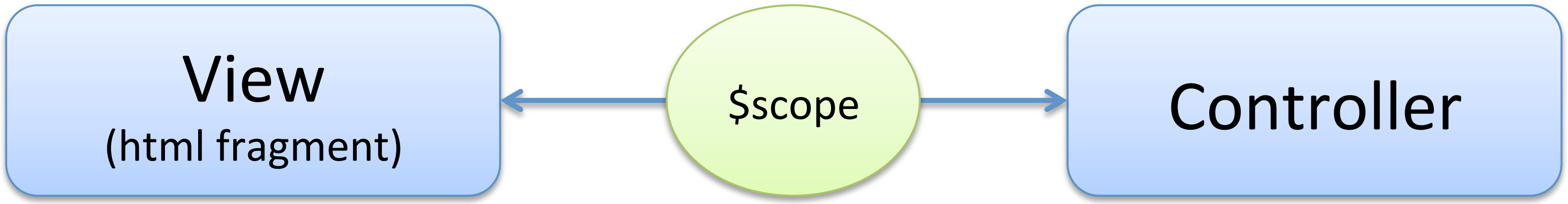
EXERCISE 1 + 2: BMI AND FILTERS

VIEWS, CONTROLLERS, SCOPE

Model – View - **Controllers**

- **Controllers** provide the **logic** behind your app.
  - So use controller when you need logic behind your UI
- AngularJS apps are controller by controllers
- Use **ng-controller** to define the controller
- Controller **is a JavaScript Object, created by** standard **JS object constructor**

View, Controller and Scope



`$scope` is an object that can *be used to communicate* between View and Controller

controller.js

```
// Angular will inject the $scope object, you don't have to
// worry about it
function NumberCtrl ($scope) {
   // $scope is bound to view, so communication
   // to view is done using the $scope
   $scope.number = 1;

   $scope.showNumber = function showNumber() {
     window.alert( "your number = " + $scope.number );
  };
}
```

Warning, this will not work from AngularJS 1.3. We will see later on how this is done using module

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
    <script src="controller.js"></script>

  </head>
  <body>
    <div data-ng-app=""
      <div data-ng-controller="NumberCtrl">
        <p>Number: <input type="number" ng-model="number"></p>
        <p>Number = {{ number }}</p>
        <button ng-click="showNumber()">Show Number</button>
      </div>
    </div>
  </body>
</html>
```

Define the Controller implemented in controller.js

Access $scope.showNumber()

Access $scope.number

App Explained

- App runs inside **ng-app** (div)
- AngularJS will invoke the constructor with a $scope – object
- $scope is an object that links controller to the view

When to use Controllers

- Use controllers
  - set up the initial state of $scope object
  - add behavior to the $scope object
- Do not
  - Manipulate DOM (use **databinding**, **directives**)
  - Format input (use **form controls**)
  - Filter output (use **filters**)
  - Share code or state (use **services**)

MODULES, ROUTES, SERVICES

Modules

- **Module** is an reusable container for different features of your app
  - **Controllers**, services, filters, directives...
- If you have a lot of controllers, you are **polluting JS namespace**
- Modules can be loaded in any order
- We can build our **own filters** and **directives**!

Example: Own Filter

```
// declare a module
var myAppModule = angular.module('myApp', []);

// configure the module.
// in this example we will create a greeting filter
myAppModule.filter('greet', function() {
 return function(name) {
    return 'Hello, ' + name + '!';
  };
});
```

HTML using the Filter

```
<div ng-app="myApp">
  <div>
    {{ 'World' | greet }}
  </div>
</div>
```

Template for Controllers

```
// Create new module 'myApp' using angular.module method.
// The module is not dependent on any other module
var myModule = angular.module('myModule',
                              []);

myModule.controller('MyCtrl', function ($scope) {
    // Your controller code here!
});
```

Creating a Controller in Module

```
var myModule = angular.module('myModule',
                              []);

myModule.controller('MyCtrl', function ($scope) {

    var model = { "firstname": "Jack",
                  "lastname": "Smith" };

    $scope.model = model;
    $scope.click = function() {
                        alert($scope.model.firstname);
                   };
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
    <script src="mymodule.js"></script>

  </head>
  <body>
    <div ng-app="myModule"
      <div ng-controller="MyCtrl">
        <p>Firstname: <input type="text" ng-model="model.firstname"></p>
        <p>Lastname: <input type="text" ng-model="model.lastname"></p>
        <p>{{model.firstname + " " + model.lastname}}</p>

        <button ng-click="click()">Show Number</button>

     </div>
    </div>
  </body>
</html>
```

This is now the model object from MyCtrl. Model object is shared with view and controller

EXERCISE 3: MODULES

ROUTING

Routing

- Since **we are building a SPA** app, everything happens in **one page**
  - How should **back-button** work?
  - How should **linking** between "pages" work?
  - How about **URLs?**
- **Routing** comes to rescue!

```
<html data-ng-app="myApp">
<head>
  <title>Demonstration of Routing - index</title>
  <meta charset="UTF-8" />
  <script src="../angular.min.js" type="text/javascript"></script>
  <script src="angular-route.min.js" type="text/javascript"></script>
  <script src="myapp.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-view=""></div>
</body>
</html>
```

We will have to load additional module

The content of this will change dynamically

```
// This module is dependent on ngRoute. Load ngRoute
// before this.
var myApp = angular.module('myApp', ['ngRoute']);

// Configure routing.
myApp.config(function($routeProvider) {
    // Usually we have different controllers for different views.
    // In this demonstration, the controller does nothing.
    $routeProvider.when('/', {
                 templateUrl: 'view1.html',
                 controller: 'MySimpleCtrl' });

    $routeProvider.when('/view2', {
                 templateUrl: 'view2.html',
                 controller: 'MySimpleCtrl' });

    $routeProvider.otherwise({ redirectTo: '/' });
});

// Let's add a new controller to MyApp
myApp.controller('MySimpleCtrl', function ($scope) {

});
```

Views

- **view1.html:**

```
<h1>View 1</h2>
<p><a href="#/view2">To View 2</a></p>
```

- **view2.html:**

```
<h1>View 2</h2>
<p><a href="#/view1">To View 1</a></p>
```

Working in Local Environment

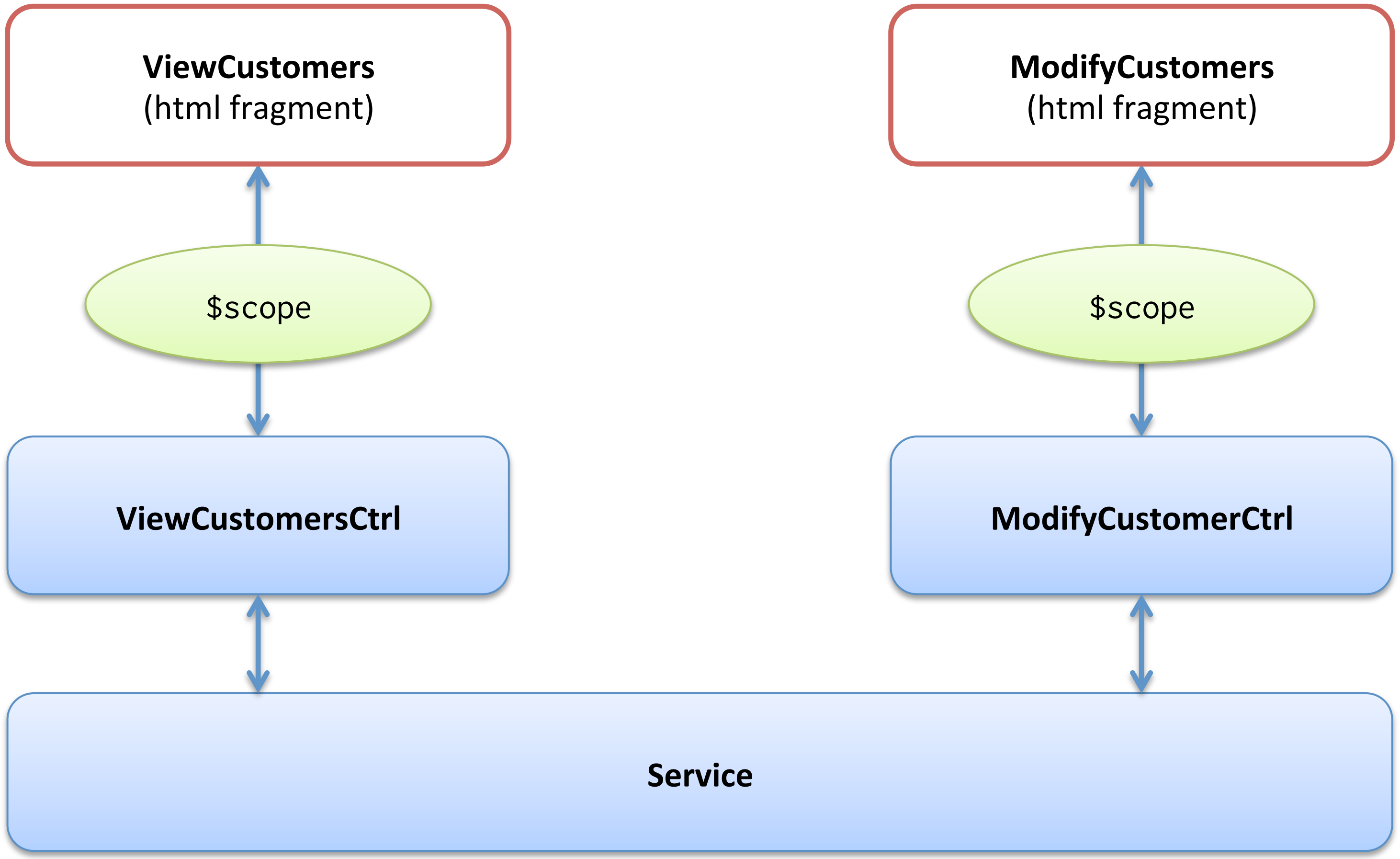
- If you get "cross origin requests are only supported for HTTP" ..
- Either
  - 1) Disable web security in your browser
  - 2) Use some web server and access files http://..
- To **disable** web security in chrome
  - `taskkill /F /IM chrome.exe`
  - `”C:\Program Files (x86)\Google\Chrome\Application\chrome.exe” --disable-web-security --allow-file-access-from-files`

EXERCISE 4: ROUTING

Services

- View-independent **business logic** should **not be** in a controller
  - Logic should be in **a service component**
- **Controllers** are **view specific**, **services** are **app-spesific**
  - We can move from view to view and service is still alive
- Controller's responsibility is to bind model to view. Model can be fetched from service!
  - Controller is not responsible for manipulating (create, destroy, update) the data. **Use Services instead!**
- AngularJS **has many built-in services**, see
  - http://docs.angularjs.org/api/ng/service
  - Example: `$http`

Services


ViewCustomers
(html fragment)
ModifyCustomers
(html fragment)
$scope
$scope
ViewCustomersCtrl
ModifyCustomerCtrl
Service

AngularJS Custom Services using Factory

```
// Let's add a new controller to MyApp. This controller uses Service!
myApp.controller('ViewCtrl', function ($scope, CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Let's add a new controller to MyApp. This controller uses Service!
myApp.controller('ModifyCtrl', function ($scope, CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Creating a factory object that contains services for the
// controllers.
myApp.factory('CustomerService', function() {
    var factory = {};
    factory.contacts = [{name: "Jack", salary: 3000}, {name: "Tina",
salary: 5000}, {name: "John", salary: 4000}];
    return factory;
});
```

Also Service

```
// Service is instantiated with new – keyword.
// Service function can use "this" and the return
// value is this.
myApp.service('CustomerService', function() {
   this.contacts =
      [{name: "Jack", salary: 3000},
       {name: "Tina", salary: 5000},
       {name: "John", salary: 4000}];
});
```

EXERCISE 5: SERVICES

AJAX + REST

AJAX

- **Asynchronous JavaScript + XML**
  – XML not needed, **very often JSON**
- Send data and retrieve asynchronously from server in background
- **Group of technologies**
  – HTML, CSS, DOM, XML/JSON, XMLHttpRequest object and JavaScript

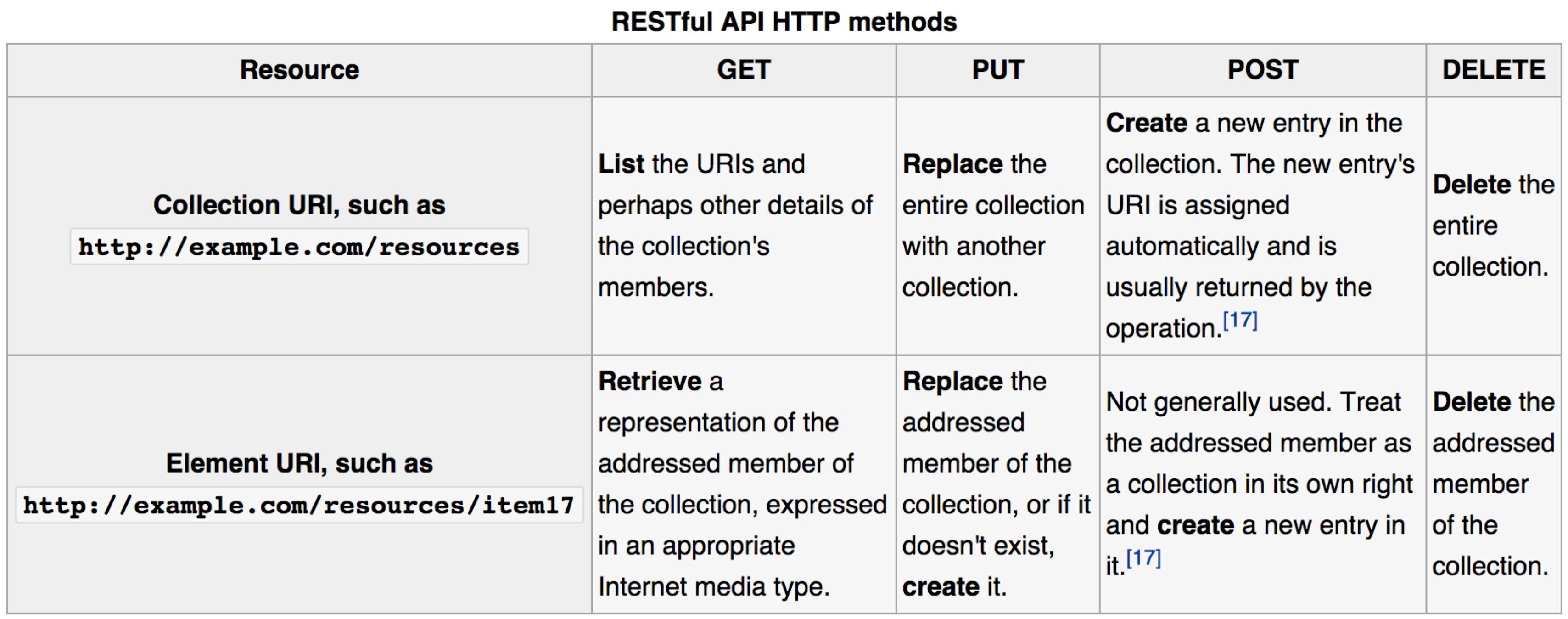
$http – example (AJAX) and AngularJS

```
<script type="text/javascript">
    var myapp = angular.module("myapp", []);

    myapp.controller("MyController", function($scope, $http) {
            $scope.myData = {};
            $scope.myData.doClick = function(item, event) {
                var responsePromise = $http.get("text.txt");

                responsePromise.success(function(data, status, headers, config) {
                    $scope.myData.fromServer = data;
                });
                responsePromise.error(function(data, status, headers, config) {
                    alert("AJAX failed!");
                });
            }
        } );
  </script>
```

RESTful

- Web Service APIs that adhere to REST architectural constrains are called RESTful
- Constrains
  - Base URI, such as http://www.example/resources
  - Internet media type for data, such as JSON or XML
  - Standard HTTP methods: GET, POST, PUT, DELETE
  - Links to reference reference state and related resources

RESTful API HTTP methods (wikipedia)



RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://example.com/resources`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[17]	**Delete** the entire collection.
Element URI, such as `http://example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[17]	**Delete** the addressed member of the collection.

AJAX + RESTful

- The web app can fetch using RESTful data from server
- Using AJAX this is done asynchronously in the background
- AJAX makes HTTP GET request using url ..
  - http://example.com/resources/item17
- .. and receives data of item17 in JSON ...
- .. which can be displayed in view (web page)

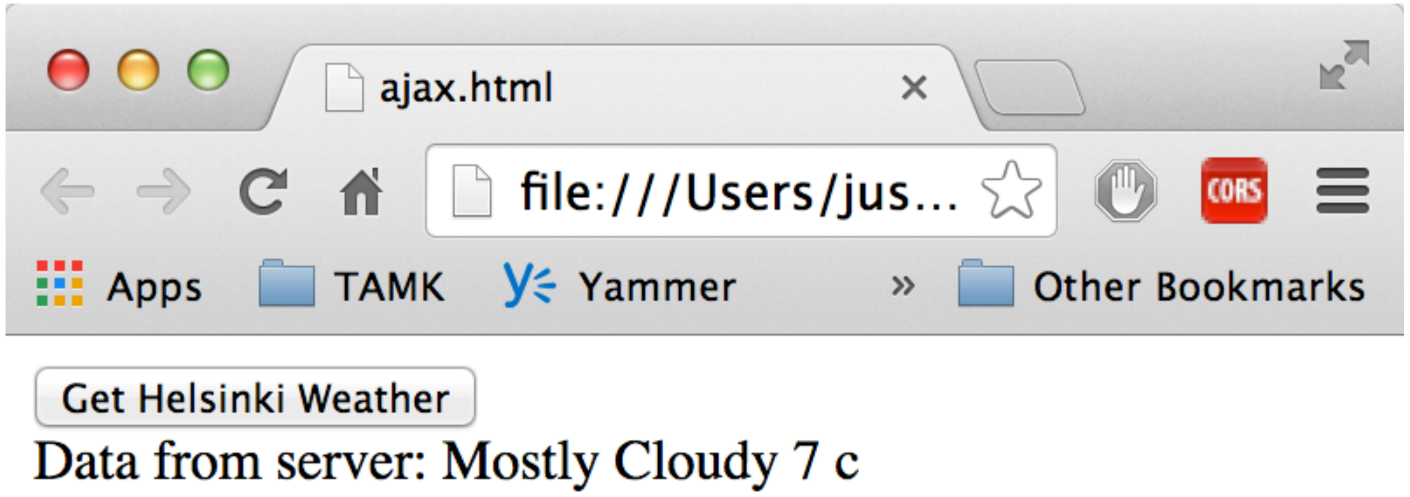
Example: Weather API

- Weather information available from `wunderground.com`
  - You have to **make account** and receive a **key**
- To get Helsinki weather in JSON
  - `http://api.wunderground.com/api/`*`your-key/`*`conditions/q/Helsinki.json`

```
{
  "response": {
    "version": "0.1",
    "termsofService": "http:\/\/www.wunderground.com\/weather\/api\/d\/terms.html",
    "features": {
      "conditions": 1
    }
  },
  "current_observation": {
    "image": {
      "url": "http:\/\/icons.wxug.com\/graphics\/wu2\/logo_130x80.png",
      "title": "Weather Underground",
      "link": "http:\/\/www.wunderground.com"
    },
    "display_location": {
      "full": "Helsinki, Finland",
      "city": "Helsinki",
      "state": "",
      "state_name": "Finland",
      "country": "FI",
      "country_iso3166": "FI",
      "zip": "00000",
      "magic": "1",
      "wmo": "02974",
      "latitude": "60.31999969",
      "longitude": "24.96999931",
      "elevation": "56.00000000"
    },
<!DOCTYPE html>
<html>
<head>
  <script src="../angular.min.js" type="text/javascript"></script>
  <title></title>
</head>

<body data-ng-app="myapp">
  <div data-ng-controller="MyController">
    <button data-ng-click="myData.doClick(item, $event)">Get Helsinki Weather</button><br />
    Data from server: {{myData.fromServer}}
  </div>

<script type="text/javascript">
    var myapp = angular.module("myapp", []);

    myapp.controller("MyController", function($scope, $http) {
            $scope.myData = {};
            $scope.myData.doClick = function(item, event) {
                var responsePromise = $http.get("http://api.wunderground.com/api/key/conditions/
q/Helsinki.json");

                responsePromise.success(function(data, status, headers, config) {
                    $scope.myData.fromServer = "" + data.current_observation.weather +
                                               " " + data.current_observation.temp_c + " c";
                });
                responsePromise.error(function(data, status, headers, config) {
                    alert("AJAX failed!");
                });
            }
        } );
  </script>
</body>
</html>
```

This is JSON object!

View after pressing the Button



Data from server: Mostly Cloudy 7 c

$resource

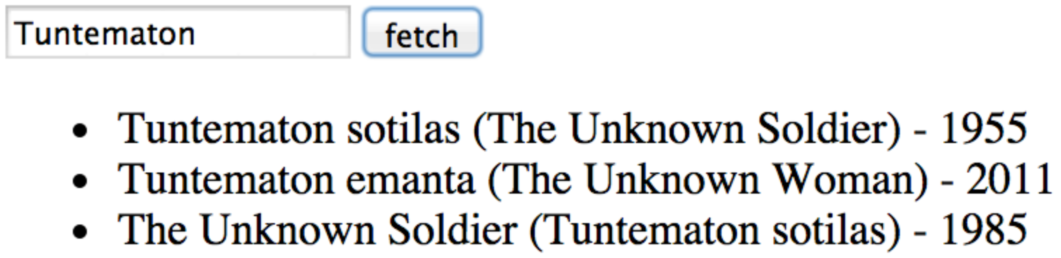
- Built on top of $http service, $resource is a factory that lets you interact with RESTful backends easily
- $resource does not come bundled with main Angular script, separately download
  - `angular-resource.min.js`
- Your main app should declare dependency on the ngResource module in order to use $resource

Getting Started with $resource

- $resource expects classic RESTful backend
  - `http://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services`
- You can create the backend by whatever technology. Even JavaScript, for example Node.js
- We are not concentrating now how to build the backend.

Using $resource on GET

```
// Load ngResource before this
var restApp = angular.module('restApp',['ngResource']);

restApp.controller("RestCtrl", function($scope, $resource) {
    $scope.doClick = function() {
            var title = $scope.movietitle;
            var searchString = 'http://api.rottentomatoes.com/api/
public/v1.0/movies.json?apikey=key&q=' + title + '&page_limit=5';

            var result = $resource(searchString);

            var root = result.get(function() {    // {method:'GET'
                $scope.movies = root.movies;
            });
        }
});
```



- Tuntematon sotilas (The Unknown Soldier) - 1955
- Tuntematon emanta (The Unknown Woman) - 2011
- The Unknown Soldier (Tuntematon sotilas) - 1985

$resource methods

- $resource contains convenient methods for
  - `get ('GET')`
  - `save ('POST')`
  - `query ('GET', isArray:true)`
  - `remove ('DELETE')`
- Calling these will invoke $http (ajax call) with the specified http method (GET, POST, DELETE), destination and parameters

Passing Parameters

```
// Load ngResource before this
var restApp = angular.module('restApp',['ngResource']);

restApp.controller("RestCtrl", function($scope, $resource) {
    $scope.doClick = function() {
            var searchString = 'http://api.rottentomatoes.com/api/public/
v1.0/movies.json?apikey=key&q=:title&page_limit=5';
            var result = $resource(searchString);
            var root = result.get({title: $scope.movietitle}, function() {
                $scope.movies = root.movies;
            });
        }
});
```

:title -> parametrized URL template

Giving the parameter from $scope

Using Services

Controller responsible for binding

```
// Load ngResource before this
var restApp = angular.module('restApp',['ngResource']);

restApp.controller("RestCtrl", function($scope, MovieService) {
    $scope.doClick = function() {
            var root = MovieService.resource.get({title: $scope.movietitle},
            function() {
                $scope.movies = root.movies;
            });
        }
});
```

Service responsible for the resource

```
restApp.factory('MovieService', function($resource) {
  factory = {};
  factory.resource = $resource('http://api.rottentomatoes...&q=:title&page_limit=5');
  return factory;
});
```

Simple Version

```
// Load ngResource before this
var restApp = angular.module('restApp',['ngResource']);

restApp.controller("RestCtrl", function($scope, MovieService) {
    $scope.doClick = function() {
            var root = MovieService.get({title: $scope.movietitle},
            function() {
                $scope.movies = root.movies;
            });
        }
});


restApp.factory('MovieService', function($resource) {
     return $resource('http://api.rottentomatoes...&q=:title&page_limit=5');;
});
```

Just call get from MovieService

Returns the resource

EXERCISE 6: REST

ANIMATIONS AND UNIT TESTING

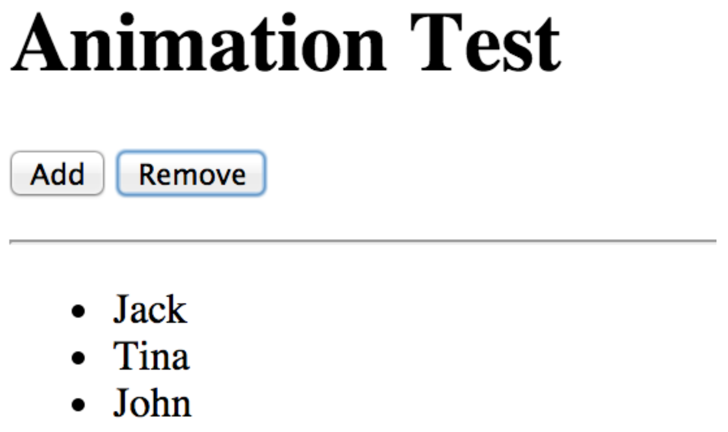
AngularJS Animations

- Include ngAnimate module as dependency
- Hook animations for common directives such as ngRepeat, ngSwitch, ngView
- Based on CSS classes
  - If HTML element has class, you can animate it
- AngularJS adds special classes to your html-elements

Example Form

```
<body ng-controller="AnimateCtrl">
  <button ng-click="add()">Add</button>
  <button ng-click="remove()">Remove</button></p>
  <ul>
    <li ng-repeat="customer in
customers">{{customer.name}}</li>
  </ul>
</body>
```

Adds and Removes names


Animation Test

- Jack
- Tina
- John

Animation Classes

- When adding a new name to the model, ng-repeat knows the item that is either added or deleted
- CSS classes are added at runtime to the repeated element (<li>)
- When adding new element:
  - `<li class=”... ng-enter ng-enter-active”>New Name</li>`
- When removing element
  - `<li class=”... ng-leave ng-leave-active”>New Name</li>`

Directives and CSS

Event	Starting CSS	Ending CSS	Directives
enter	.ng-enter	.ng-enter-active	ngRepeat, ngInclude, ngIf, ngView
leave	.ng-leave	.ng-leave-active	ngRepeat, ngInclude, ngIf, ngView
move	.ng-move	.ng-move.active	ngRepeat

Example CSS

```
/* starting animation */
.ng-enter {
  -webkit-transition: 1s;
  transition: 1s;
  margin-left: 100%;
}

/* ending animation */
.ng-enter-active {
  margin-left: 0;
}

/* starting animation */
.ng-leave {
  -webkit-transition: 1s;
  transition: 1s;
  margin-left: 0;
}

/* ending animation */
.ng-leave-active {
  margin-left: 100%;
}
```

Test Driven Design

- Write tests firsts, then your code
- AngularJS emphasizes modularity, so it can be easy to test your code
- Code can be tested using several unit testing frameworks, like QUnit, Jasmine, Mocha ...

QUnit

- Download `qunit.js` and `qunit.css`
- Write a simple HTML page to run the tests
- Write the tests

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="qunit-1.10.0.css">
  <script src="qunit-1.10.0.js"></script>
</head>
<body>
  <div id="qunit"></div>
  <script type="text/javascript">

    function calculate(a, b) {
        return a + b;
    }

    test( "calculate test", function() {
      ok( calculate(5,5) === 10, "Ok!" );
      ok( calculate(5,0)  === 5, "Ok!" );
      ok( calculate(-5,5) === 0, "OK!" );
    });

  </script>
</body>
</html>
```

Three Assertions

- Basic
  - `ok( boolean [, message]);`
- If *actual == expected*
  - `equal( actual, expected [, message]);`
- if *actual === expected*
  - `deepEqual( actual, expected [, message));`
- Other
  - `http://qunitjs.com/cookbook/#automating-unit-testing`

Testing AngularJS Service

```
var myApp = angular.module('myApp', []);

// One service
myApp.service('MyService', function() {
    this.add = function(a, b) {
        return a + b;
    };
});

/* TESTS */
var injector = angular.injector(['ng', 'myApp']);

QUnit.test('MyService', function() {
    var MyService = injector.get('MyService');
    ok(2 == MyService.add(1, 1));
});
```

EXERCISE 7: QUNIT AND ANGULAR_JS

WRAPPING UP

Wrapping UP

- AngularJS is a modular JavaScript SPA framework
- Lot of great features, but learning curve can be hard
- Great for CRUD (create, read, update, delete) apps, but not suitable for every type of apps
- Works very well with some JS libraries (JQuery)