



Intro to SPA framework

Based on a presentation by
Jussi Pohjolainen

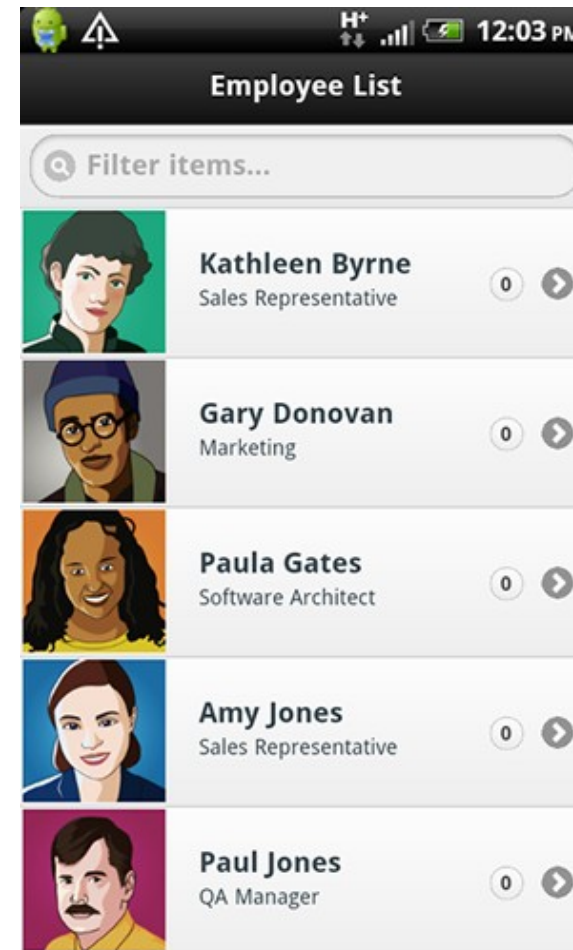
Rise of the Responsive Single Page App



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

Responsive

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



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,
- Requires handling of **browser history, navigation and bookmarks**

JavaScript

- SPAs are implemented using **JavaScript** and **HTML**

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 ...**

Single-page Application

Single page apps typically have

- “application like” interaction
- dynamic data loading from the server-side API
- fluid transitions between page states
- more JavaScript than actual HTML

They typically do not have

- support for crawlers (not for sites relying on search traffic)
- support for legacy browsers (IE7 or older, dumbphone browsers)

SPAs Are Good For ...

- “App-like user experience”
- Binding to your own (or 3rd party) RESTful API
- Replacement for Flash or Java in your web pages
- Hybrid (native) HTML5 applications
- Mobile version of your web site

*The SPA sweet spot is likely not on web sites,
but on content-rich cross-platform mobile apps*

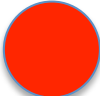
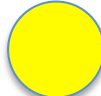
PJAX

Pjax is a technique that allows you to progressively enhance normal links on a page so that clicks result in the linked content being loaded via Ajax and the URL being updated using HTML5 pushState, avoiding a full page load.

In browsers that don't support pushState or that have JavaScript disabled, link clicks will result in a normal full page load. The Pjax Utility makes it easy to add this functionality to existing pages.

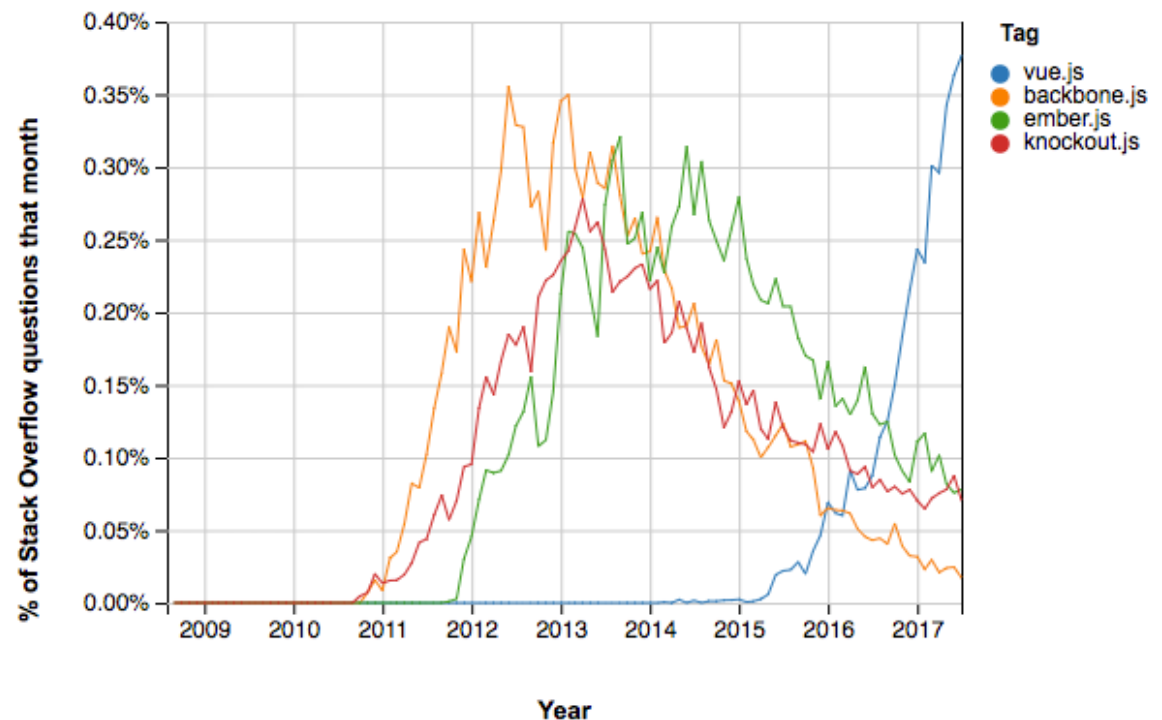
<http://yuilibrary.com/yui/docs/pjax/>

SPAs and Other Web App Architectures

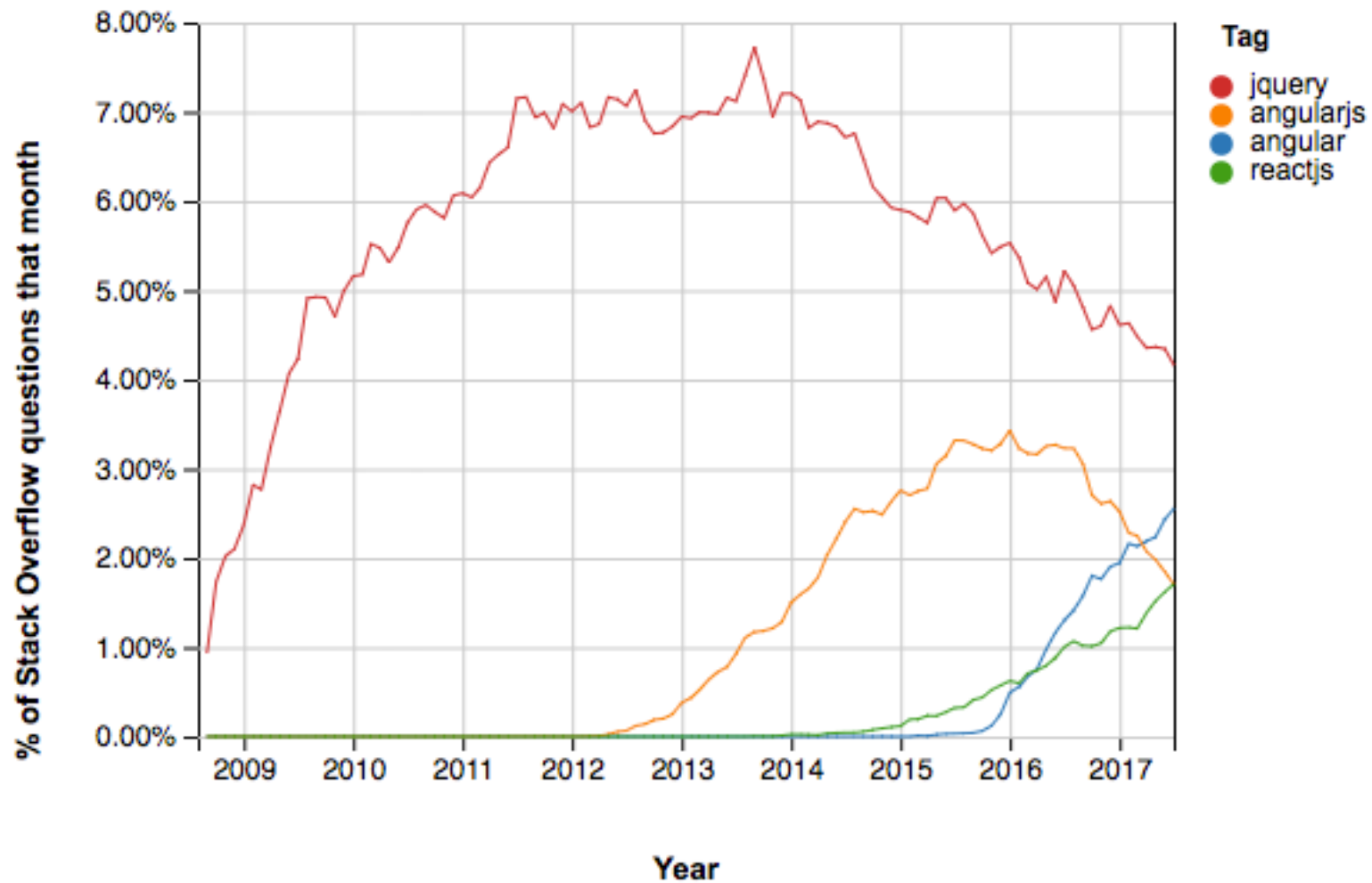
	Server-side	Server-side + AJAX	PJAX	SPA
What	Server round-trip on every app state change	Render initial page on server, state changes on the client	Render initial page on server, state changes on server, inject into DOM on client-side	Serve static page skeleton from server; render every change on client-side
How	UI code on server; links & form posting	UI code on both ends; AJAX calls, ugly server API	UI code on server, client to inject HTTP, server API if you like	UI code on client, server API
Ease of development				
UX & responsiveness				
Robots & old browsers				
Who's using it?	Amazon, Wikipedia; banks, media sites etc.	Facebook?; widgets, search	Twitter, Basecamp, GitHub	Google+, Gmail, FT; mobile sites, startups

Lifecycle of new JS frameworks

There appears to be a **quick ascent**, as the framework gains popularity and then a slightly less quick but **steady decline** as developers adopt newer technologies. These lifecycles only last **a couple of years**.



Jquery, Angular JS, Angular, React



ANGULAR_JS

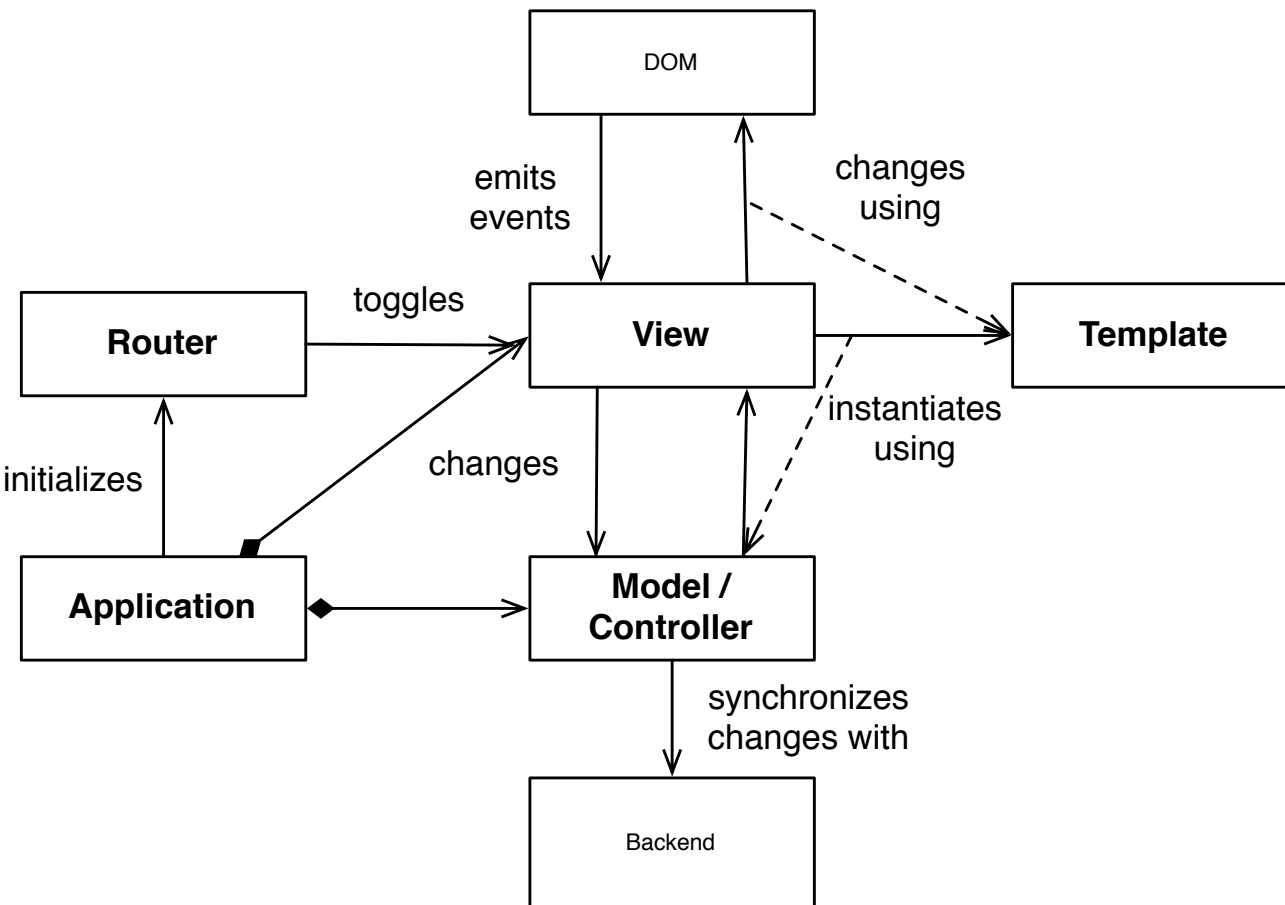
Angular JS

- **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

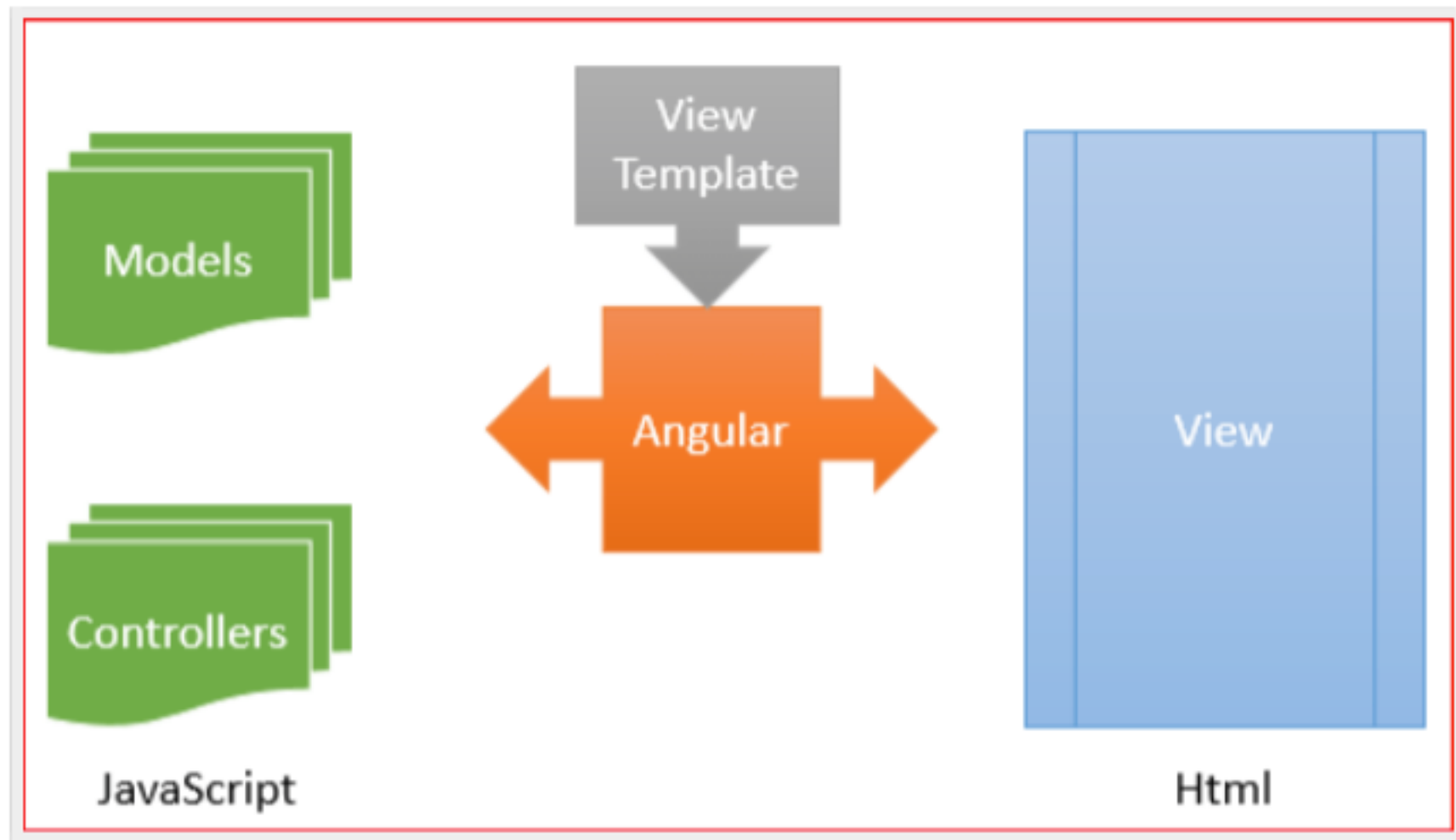
AngularJS – Main Concepts

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

Anatomy of a Backbone SPA

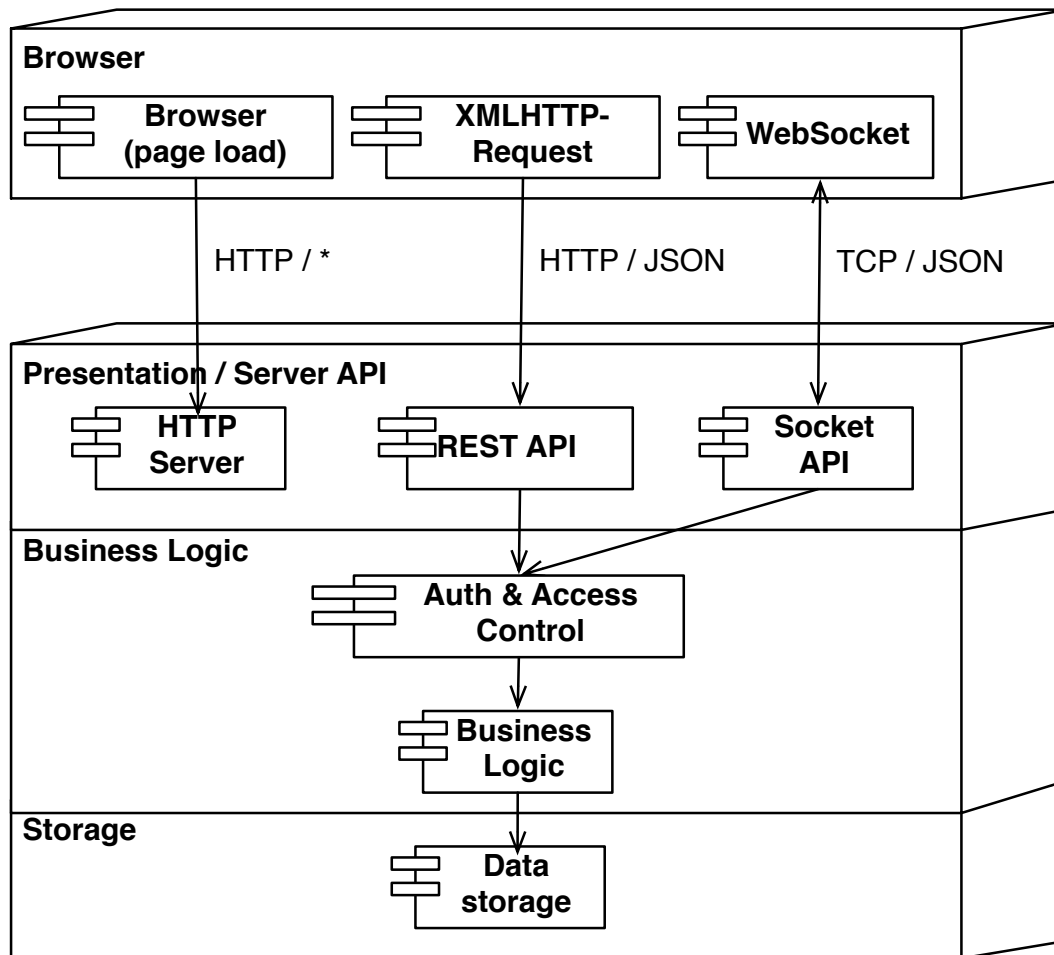


- Application as a 'singleton' reference holder
- Router handles the navigation and toggles between views
- Models synchronize with Server API
- Bulk of the code in views
- All HTML in templates



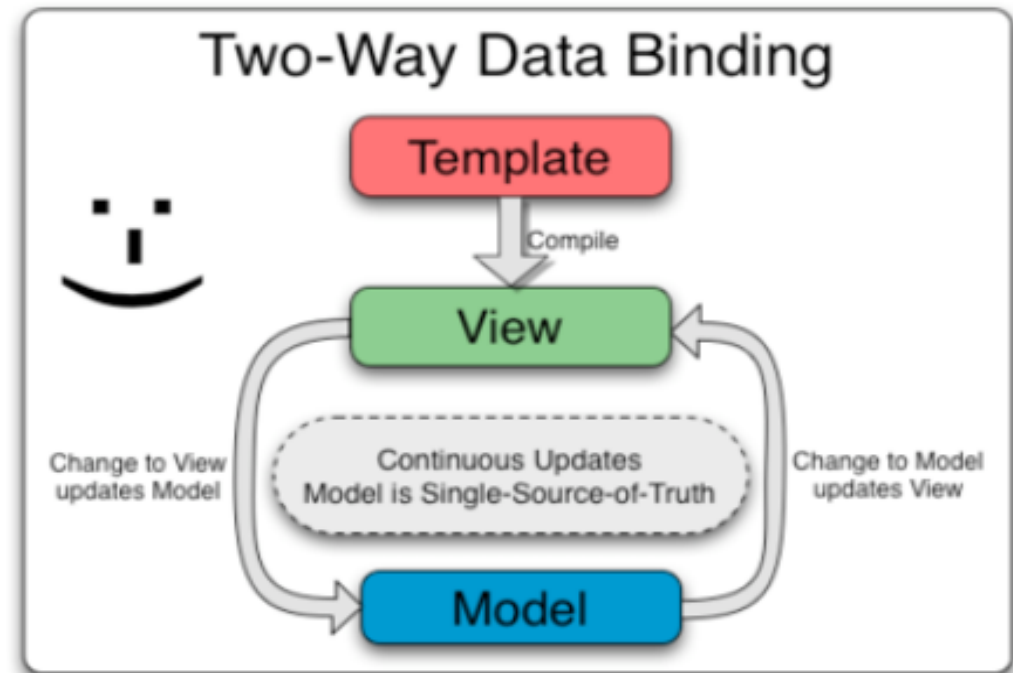
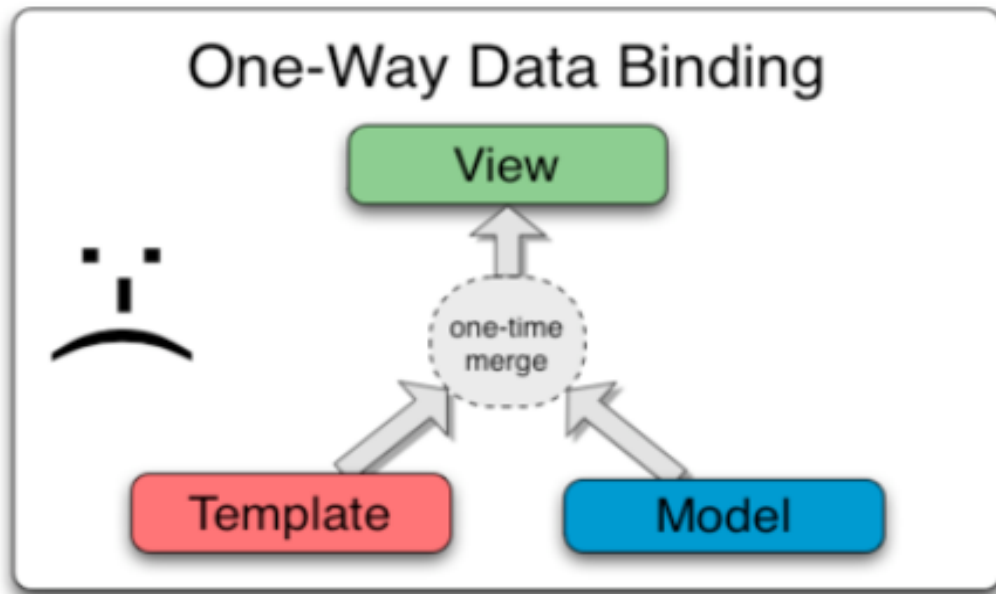
From Gary Arora

SPA Client-Server Communication

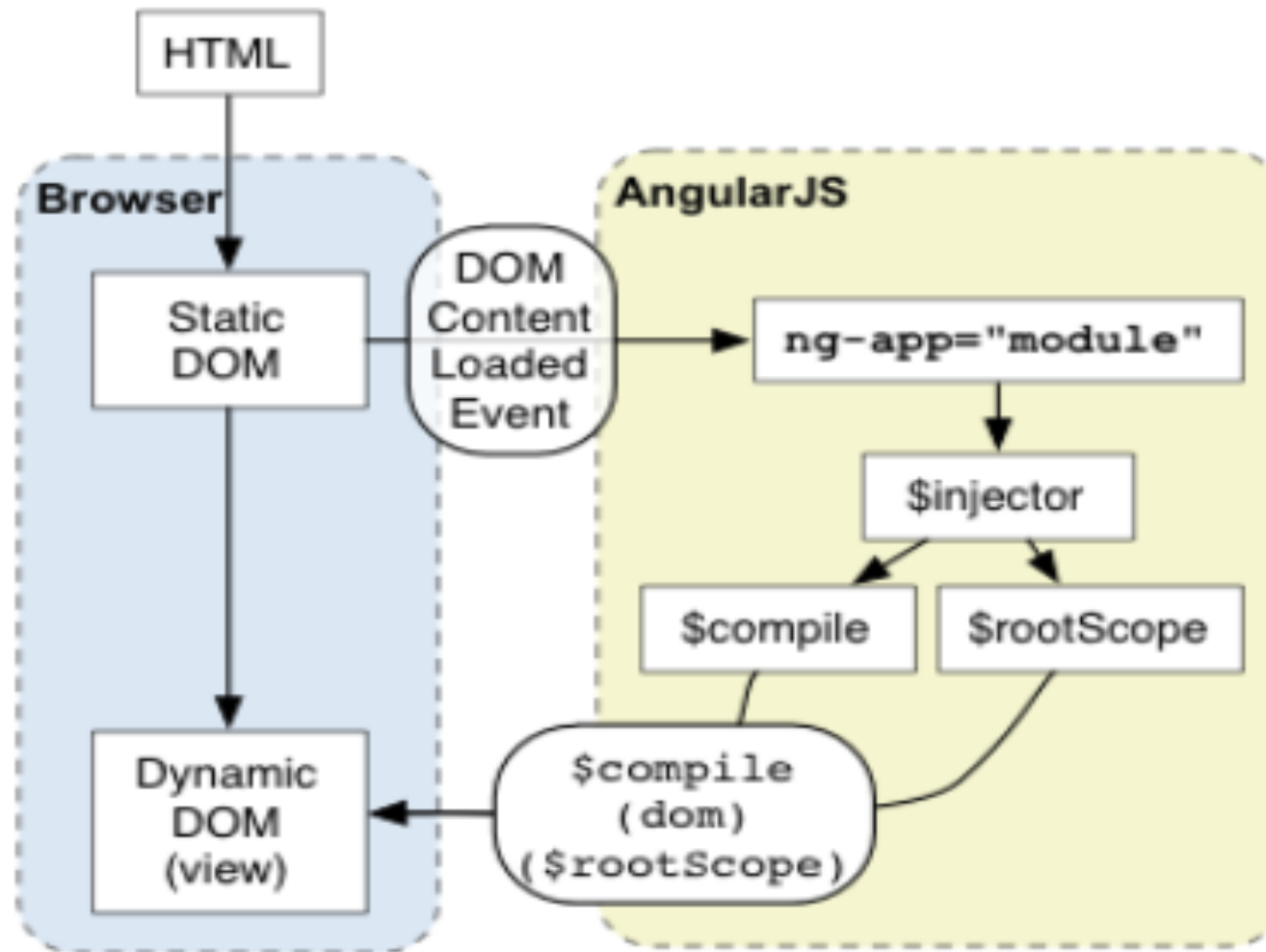


- HTML and all the assets are loaded in first request
- Additional data is fetched over XMLHttpRequest
- If you want to go real-time, WebSockets (socket.io) can help you
- When it gets slow, cluster the backend behind a caching reverse proxy like [Varnish](https://varnish.org/)

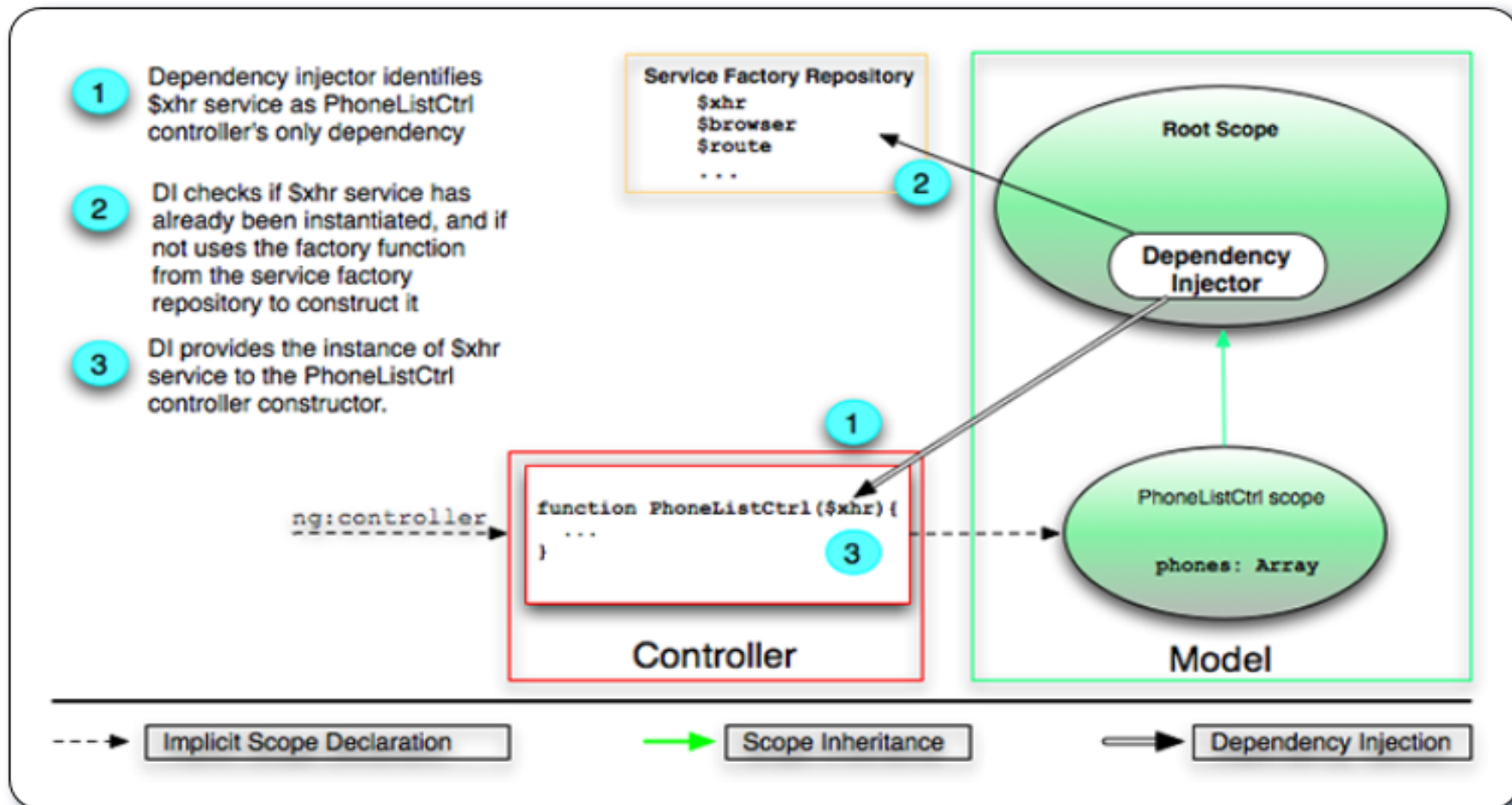
HOW IT WORKS?



HOW IT WORKS?



HOW IT WORKS?



GETTING STARTED WITH ANGULAR_JS

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 `{{ }}`

Name:

pippo

First Example – Template

Template

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/
1.4.8/angular.min.js"></script>
  </head>
  <body>
    <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>
```

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`
- `ngDb1Click`
- `ngMouseEnter`
- `ngMouseMove`
- `ngMouseLeave`**e**
- `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

3) Filter

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

Directives

The ng-app Directive

The ng-app directive defines the root element of an AngularJS application. It will auto-bootstrap (automatically initialize) the application when a web page is loaded.

The ng-init Directive

The ng-init directive defines initial values for an AngularJS application.

The ng-model Directive

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-bind Directive

The ng-model directive binds the innerHTML of the element to the specified model property

Directives: ng-app, ng-model, ng-bind

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

Directives: ng-init, and version for valid HTML

```
<div ng-app="" ng-init="firstName='John'">
```

```
<p>The name is <span ng-bind="firstName"></span></p>
```

```
</div>
```

```
<div ng-app="" data-ng-init="firstName='John'">
```

```
<p>The name is <span data-ng-bind="firstName"></span></p>
```

```
</div>
```

Form input

Checkbox => true, false

Radio => values

Select => values

Text => value

```
<form>
```

Check to show a header:

```
<input type="checkbox" ng-model="myVar">
```

```
</form>
```

```
<h1 ng-show="myVar">My Header</h1>
```

```
<select ng-model="myVar">
```

```
<option value="">
```

```
<option value="dogs">Dogs
```

```
<option value="tuts">Tutorials
```

```
<option value="cars">Cars
```

```
</select>
```

```
<input type="text" ng-model="user.firstName">
```

```
<input type="radio" ng-model="myVar" value="dogs">Dogs</input>
```

```
<input type="radio" ng-model="myVar" value="tuts">Tutorials</input>
```

```
<input type="radio" ng-model="myVar" value="cars">Cars</input>
```

Input field properties

Input fields have the following states:

\$untouched The field has not been touched yet

\$touched The field has been touched

\$pristine The field has not been modified yet

\$dirty The field has been modified

\$invalid The field content is not valid

\$valid The field content is valid

Values: either true or false.

Show an error message if the field has been touched AND is empty:

```
<input name="myName" ng-model="myName" required>  
<span ng-show="myForm.myName.$touched &&  
myForm.myName.$invalid">The name is required.</span>
```

Forms have the following states:

\$pristine No fields have been modified yet

\$dirty One or more have been modified

\$invalid The form content is not valid

\$valid The form content is valid

\$submitted The form is submitted

Values: either true or false.

Basic validation

```
<form name="myForm">  
  <input name="myInput"  
    ng-model="myInput" required>  
</form>
```

```
<p>The input's valid state is:</p>  
<h1>{{myForm.myInput.$valid}}</h1>
```

```
<form name="myForm">  
  <input name="myInput" ng-model="myInput" type="email">  
</form>
```

```
<p>The input's valid state is:</p>  
<h1>{{myForm.myInput.$valid}}</h1>
```

Directives for DOM control

```
<p ng-show="false">I am not visible.</p>  
<button ng-disabled="mySwitch">Click Me!</button>
```



Control this variable
via code

Expressions

```
<!DOCTYPE html>
<html lang="en-US">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

  <div ng-app="">
    <p>Name : <input type="text" ng-model="name"></p>
    <h1>Hello {{name}}</h1>
  </div>

</body>
</html>
```

Angular JS vs plain JS

Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.

Unlike JavaScript expressions, AngularJS expressions can be written inside HTML.

AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.

AngularJS expressions support filters, while JavaScript expressions do not

Expressions

```
<div ng-app="" ng-init="myCol='lightblue'">
```

```
<input style="background-color:{{myCol}}" ng-model="myCol">
```

```
</div>
```

Change the value of the input field:

A rectangular input field with a red background and a thin grey border. The word "red" is written in black text inside the field.

```
<div>{{5*8}}</div>
```


Directives: ng-repeat

The ng-repeat directive clones HTML elements once for each item in a collection.

The ng-repeat directive is used on an array of objects:

```
<div ng-app="" ng-init="names=[  
  {name:'Jani',country:'Norway'},  
  {name:'Hege',country:'Sweden'},  
  {name:'Kai',country:'Denmark'}]">
```

```
<ul>  
  <li ng-repeat="x in names">  
    {{ x.name + ', ' + x.country }}  
  </li>  
</ul>
```

```
</div>
```

Filters

AngularJS provides filters to transform data:

currency	Format a number to a currency format.
date	Format a date to a specified format.
number	Format a number to a string.
filter	Select a subset of items from an array.
json	Format an object to a JSON string.
lowercase	Format a string to lower case.
uppercase	Format a string to upper case.
orderBy	Orders an array by an expression.
limitTo	Limits an array/string, into a specified number of elements/characters.

Filters

Filters can be added to expressions by using the pipe character |, followed by a filter.

```
<div ng-app="myApp" ng-init="lastName='ronchetti'">
```

```
<p>The name is {{ lastName | uppercase }}</p>
```

```
</div>
```

API Reference

<https://docs.angularjs.org/api/ng/filter/filter>

The screenshot shows a web browser window displaying the AngularJS API Reference for the 'filter' module. The browser's address bar shows the URL <https://docs.angularjs.org/api/ng/filter/filter>. The page has a dark navigation bar with the AngularJS logo and links for Home, Learn, Develop, and Discuss. A search bar is also present. Below the navigation bar, a breadcrumb trail indicates the current location: v1.3.0-build.3422 (snapshot) / API Reference / ng / filter components in ng / filter. On the left side, a sidebar lists various filter categories: filter (selected), currency, date, json, limitTo, lowercase, number, orderBy, and uppercase. Below these are sections for 'auto', 'service' (with \$injector and \$provide), 'ngAnimate' (with \$animateProvider), 'provider' (with \$ariaProvider), and 'ngAria' (with \$aria). The main content area is titled 'filter' and includes a subtitle '- filter in module ng'. It contains a description: 'Selects a subset of items from array and returns it as a new array.' There are two buttons: 'View Source' and 'Improve this Doc'. The 'Usage' section is divided into 'In HTML Template Binding' and 'In JavaScript'. The HTML binding example shows the syntax: `{{ filter_expression | filter : expression : comparator }}`. The JavaScript example shows the function call: `$filter('filter')(array, expression, comparator)`. The 'Arguments' section contains a table with three columns: Param, Type, and Details.

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:

More examples

Example

Number 1:

Number 2:

13

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
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

ng-init and ng-repeat directives

Cool loop!

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular.min.js"></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>
```

- Jack
- John
- Tina

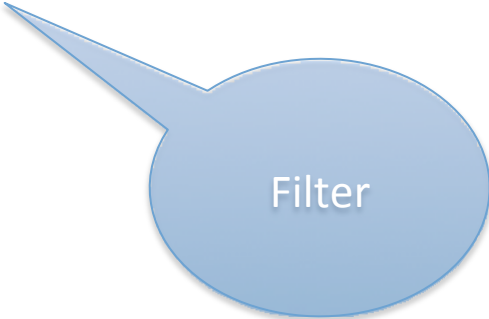
Cool loop!

Using Filters –Example

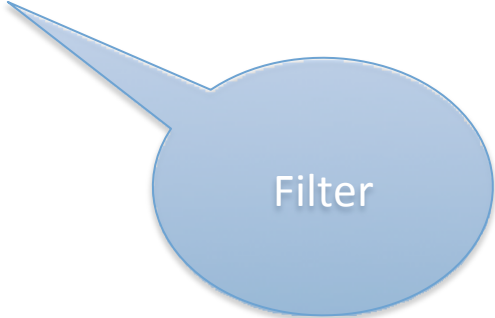
- JACK
- TINA

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular.min.js"></script>
  </head>
  <body>
    <div data-ng-init="customers = [{name:'tina'}, {name:'jack'}]">
      <h1>Cool loop!</h1>
      <ul>
        <li data-ng-repeat="customer in customers | orderBy:'name'">
          {{ customer.name | uppercase }}</li>
        </ul>
      </div>
    </body>

  </html>
```



Filter



Filter

Using Filters –Example

Customers

- JOHN

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></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

Customers

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular.min.js"></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>
```

- JACK
- JOHN

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>
```

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 controlled by controllers
- Use **ng--controller** to define the controller
- Controller is a **JavaScript Object**, created by standard **JS object constructor**

Model – View –**Controllers**

a controller is a JavaScript function

- It contains data
- It specifies the behavior
- It should contain only the business logic needed for a single view.

Modules

A module is created by using the AngularJS function `angular.module`. To it you can add components (your own controllers, directives, filters...)

```
<div ng-app="myApp">...</div>
```

```
<script>
```

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

```
</script>
```

Controllers

A simple AngularJS application to consist of:

- View, which is the HTML.
- Model, which is the data available for the current view (scope)
- Controller, which is the JavaScript function that makes/changes/removes/controls the data.

Controllers

AngularJS applications are controlled by controllers.

The ng-controller directive defines the application controller.

A controller is a JavaScript Object, created by a standard JavaScript object constructor.

Controllers

```
<div ng-app="myApp" ng-controller="myCtrl">
```

```
First Name: <input type="text" ng-model="firstName"><br>
```

```
Last Name: <input type="text" ng-model="lastName"><br>
```

```
<br>
```

```
Full Name: {{firstName + " " + lastName}}
```

```
</div>
```

```
<script>
```

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

```
app.controller('myCtrl', function($scope) {
```

```
    $scope.firstName = "John";
```

```
    $scope.lastName = "Doe";
```

```
});
```

```
</script>
```

- Add a controller which implements some logic
- Define variables in app space
- Access variables from controller by using \$scope

Local and global scope

```
<body ng-app="myApp">
<p>The rootScope's favorite color:</p>
<h1>{{color}}</h1>
```

```
<div ng-controller="myCtrl">
  <p>The scope of the controller's favorite color:</p>
  <h1>{{color}}</h1>
</div>
```

```
<p>The rootScope's favorite color is still:</p>
<h1>{{color}}</h1>
```

All applications have a `$rootScope` which is the scope created on the HTML element that contains the ng-app directive.

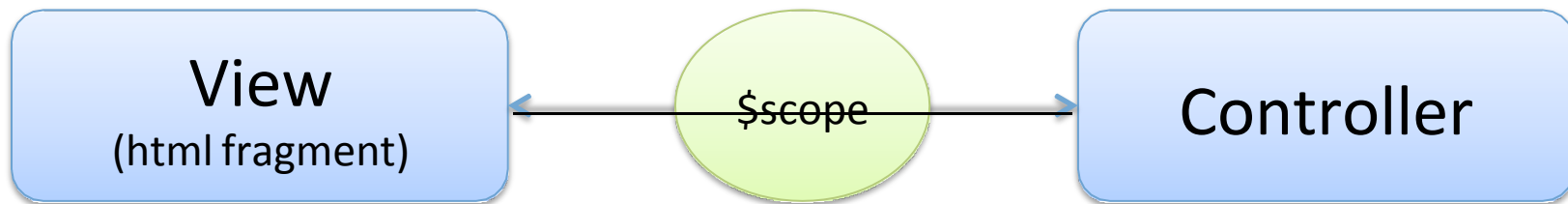
```
<script>
var app = angular.module('myApp', []);

app.run(function($rootScope) {
  $rootScope.color = 'blue';
});

app.controller('myCtrl', function($scope) {
  $scope.color = "red";
});

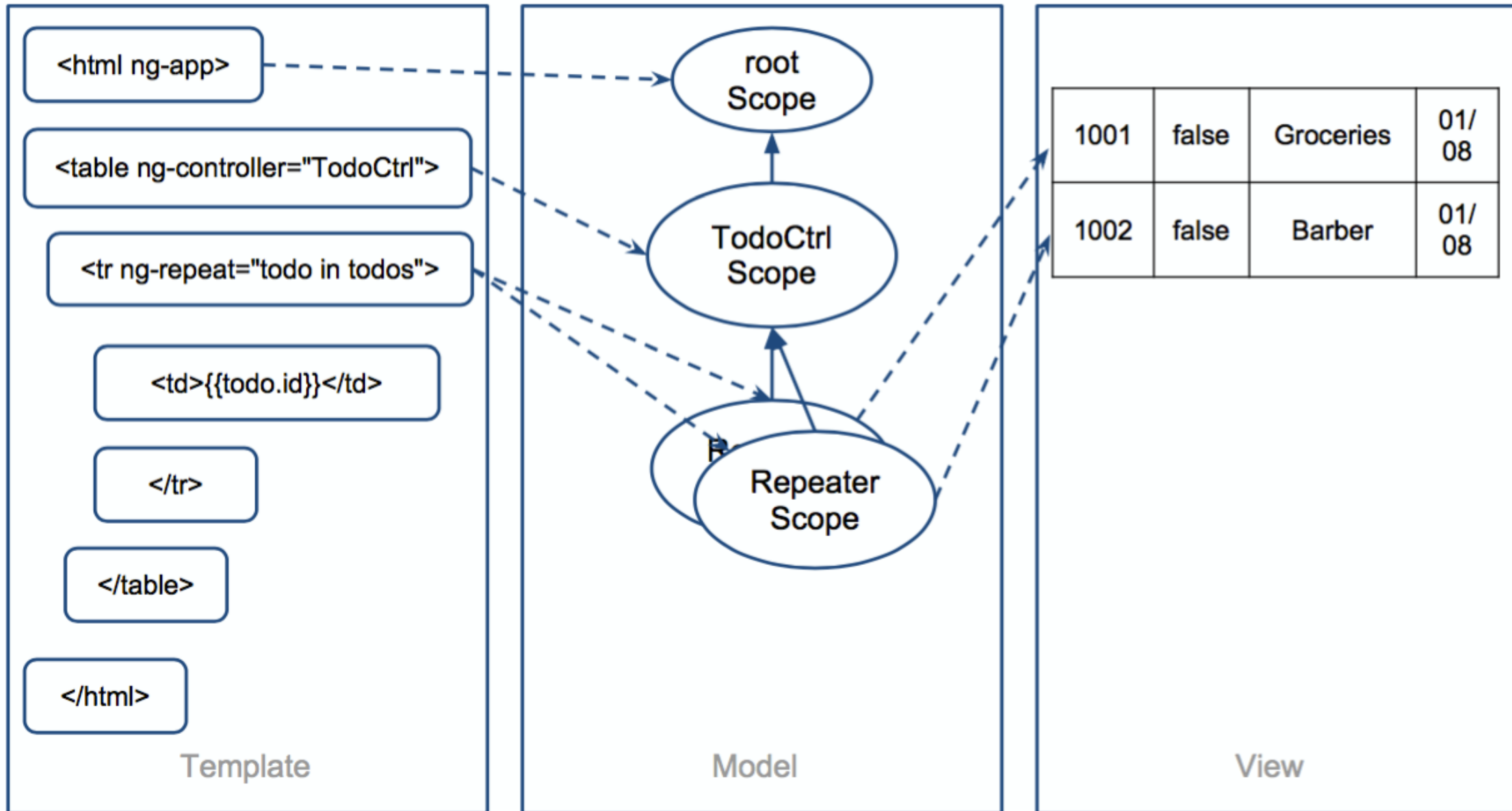
</script>
</body>
```

View, Controller and Scope



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

Scope

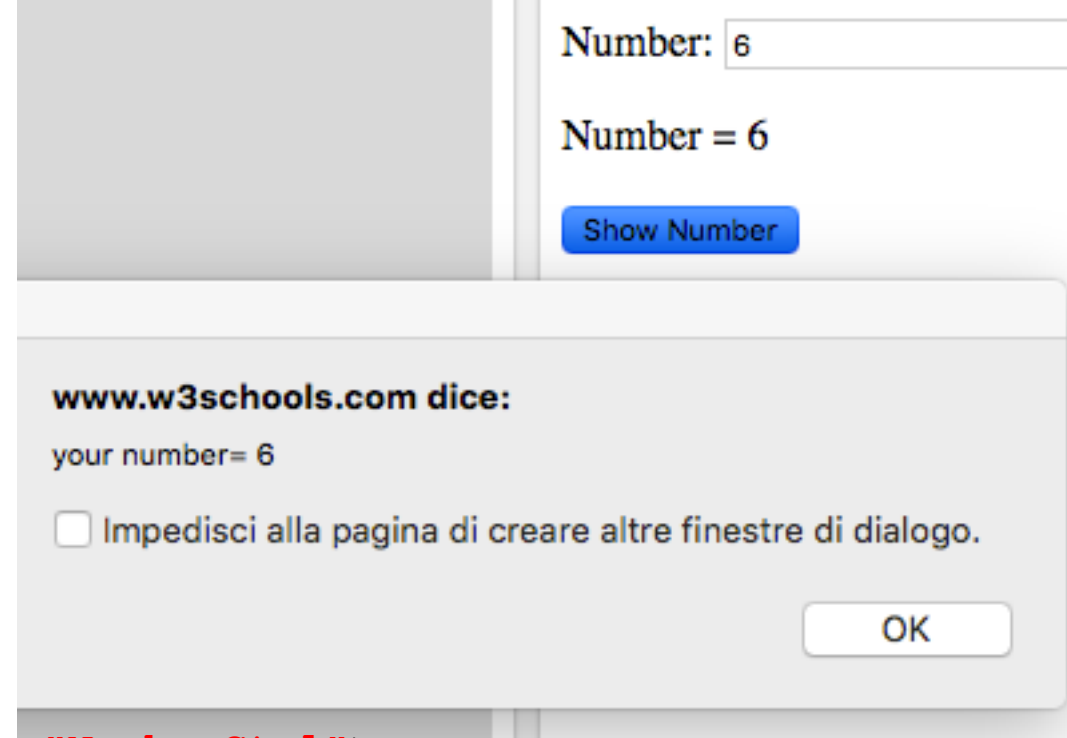


```

<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/
ajax/libs/angularjs/1.4.8/angular.min.js">
</script>

  </head>
  <body>
<div data-ng-app="myApp" 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>
<script>
var app = angular.module('myApp', []);
app.controller('NumberCtrl', function($scope) {
  $scope.number = 1;
  $scope.showNumber = function(){
    window.alert( "your number= " + $scope.number );
  };
});
</script>
</body>
</html>

```



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!**

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**)

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

MODULES, ROUTES, SERVICES

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

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

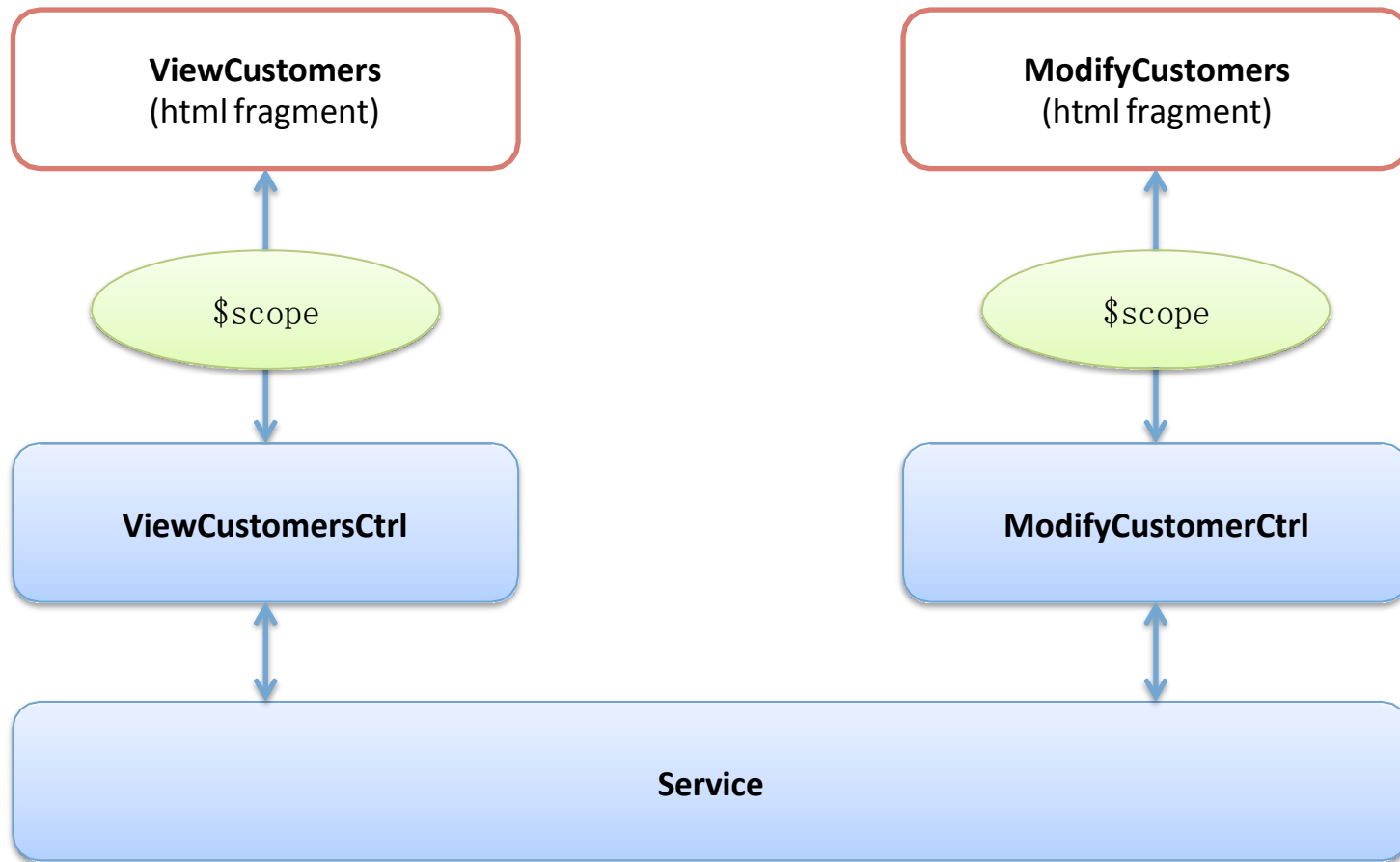
- If you get "cross origin requests are only supported for HTTP" ..
- Either
 - 1) Disable web security in your browser
 - 2) Use some [web server and access files http://..](http://..)
- **To disable web security in chrome**
 - `taskkill /F /IM chrome.exe`
 - `"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"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



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}];  
    });
```

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 constraints are called RESTful
- Constraints
 - 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 <code>http://example.com/resources</code>	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 <code>http://example.com/resources/item17</code>	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



Get Helsinki Weather

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

```
// 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;
        });
    }
});

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

Controller
responsible for
binding

Service
responsible for
the resource

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

ANIMATIONS AND UNIT TESTING

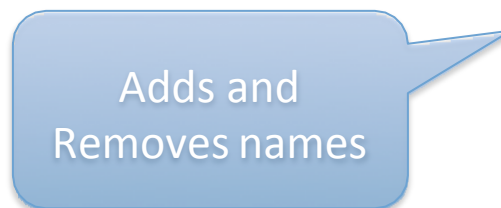
AngularJS Animations

- 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>
```

Animation Test



Add Remove

-
- 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 ()
- 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

```
<!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));
});
```

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)

Services

The AngularJS `$http` service makes a request to the server, and returns a response.

```
app.controller('myCtrl', function($scope, $http) {  
    $http.get("welcome.htm")  
    .then(function(response) {  
        $scope.myWelcome = response.data;  
    });  
});
```

