Introduction to AngularJS

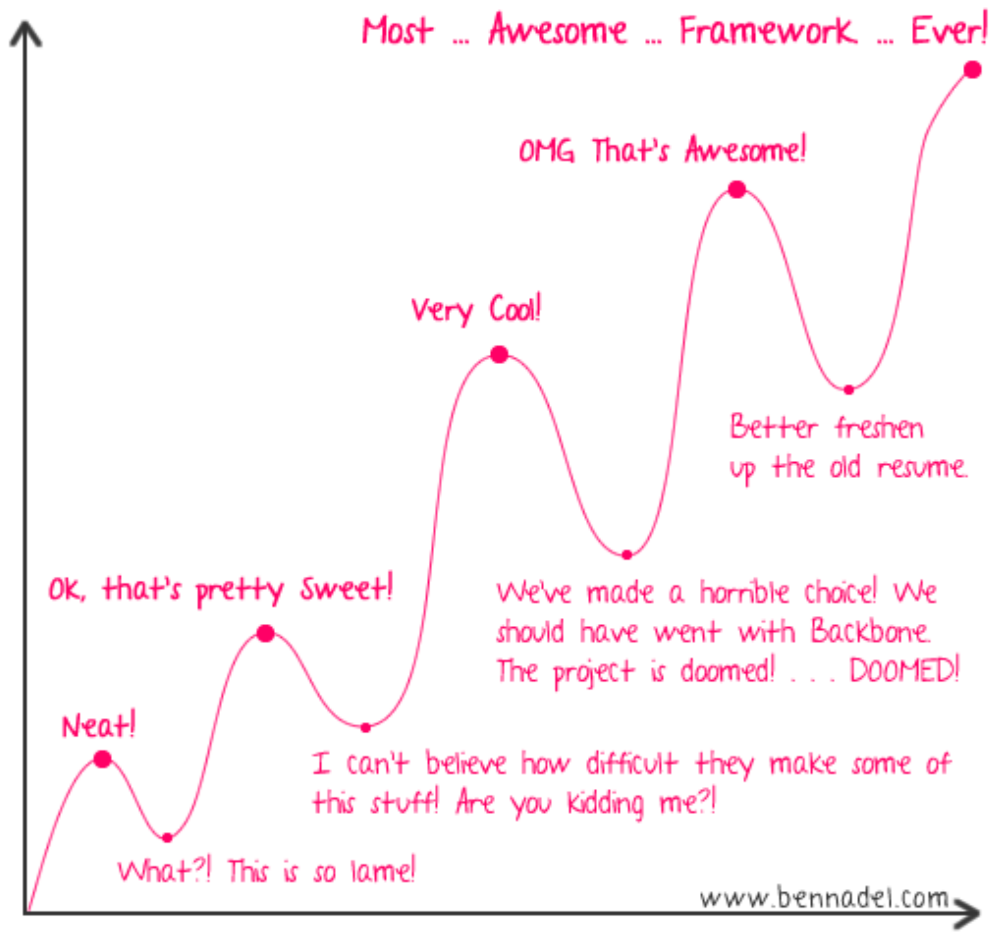
George Azmy | @grgzmy | grg@azmy.ca

AngularJS

its not a ~~Library~~

its kind of a Framework

but really an HTML Extender (or compiler)



My Feelings About **AngularJS** Over Time

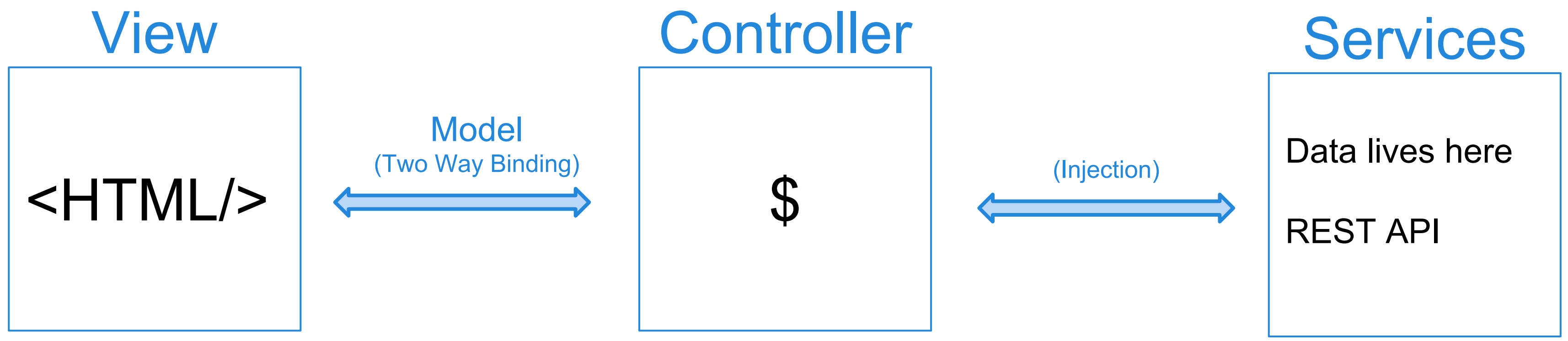
Design Pattern

MVC $\longrightarrow$

MVVM $\longrightarrow$

MVWhateverWorksForYou

Structure


View
<HTML/>
Model
(Two Way Binding)
Controller
$
(Injection)
Services
Data lives here
REST API

View: Bye-Bye DOM manipulations

Angular introduces many “directives” that do all your DOM manipulations

They turn HTML from less of a descriptive language to more of a programming language

Wait..directives?

Custom Angular tags and attributes that do all your magic.

Normalized

from camelCase (in js)

to hyphen-case (in HTML)

Views: HTML logic

ng-if : adds/removes elements from DOM

```
<p ng-if="price > 0">BUY</p>
```

ng-show/ng-hide: CSS show/hide elements

```
<p ng-show="price < 0">WARN</p>
<p ng-hide="price > 0">YIKES</p>
```

View: Dynamic styles

ng-class: Dynamically change css class

```
<p ng-class="{red:price<0,green:price>5}">
{{price}}</p>
```

```
<p ng-class= "price < 0 ? 'red' : 'green'">
{{price}}</p>
```

ng-style: Similar to ng-class but adds styles

View: Loops

ng-repeat: repeats any element based on number of elements in array, or KVPs in objects

```
<p ng-repeat= “pug in pugs”>{{pug.name}}</p>
```

```
<p ng-repeat =“(key, value) in pug”>{{key}} is {{value}}</p>
```

```
<li ng-repeat=
“pug in pugs”>
{{pug.name}}
</li>
```

- alex
- ashley
- robin

```
<li ng-repeat=
“(key, value)
in pug”>
{{key}} is
{{value}}
</li>
```

- name is pablo
- age is 12
- owner is @grgzmy

View: Filters

Pipe data into $filters to format/manipulate, eg

- Currency
- Date
- Time
- Filtration (to search arrays)
- Sort

```
<p> {{startDate | date:dd/MM/yy}}</p>
```

```
pugFi = "a"
```

```
<li ng-repeat=
"pug in pugs|
filter:pugFi">
{{pug.name}}
</li>
```

→

- alex
- ashley

The Model

Binds the View to the HTML

Can be one way, once, or two way

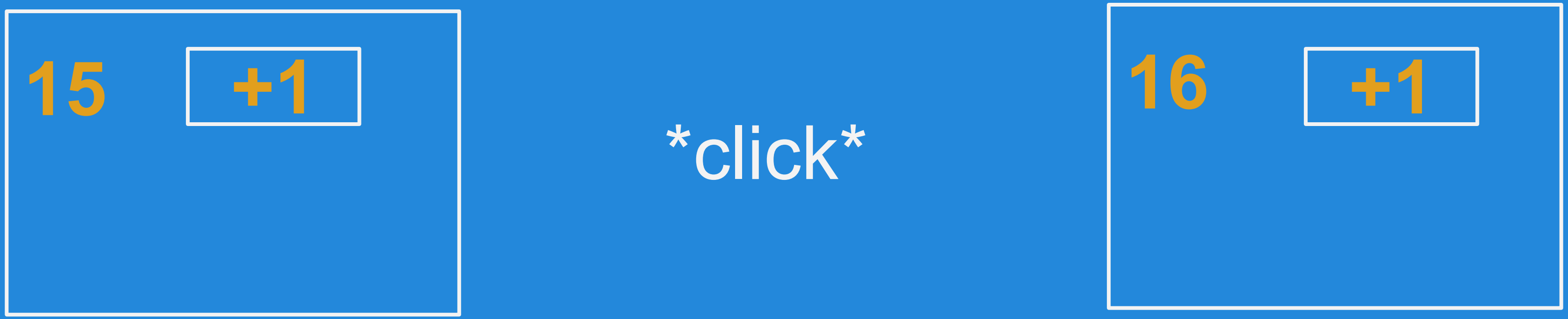
Updates in View reflect in Controller and vice-versa

```
<span>Hi: {{myInput}}</span>
<input ng-model=”myInput”/>
```

Hi: pugs pugs

pug pugs

```
<span>Age:{{age}}</span>
<button ng-click="growUp()">
   +1
</button>
```



The Controller

Is a **function**

Declares a **$scope** for the view, aka the **Model**

Enriches **$scope** with **functions** and **data**

The Controller

Hierarchical: Parent-Children inheritance (and subsequent $scopes)

View can read from parent $scopes upwards

Parent scope is the $rootScope, created for every Angular app

The controller

Should not hold the data

Should hold logic+flow functions or variables to maintain state

Should hook up minimum required data from services (data layer) to $scope

The $scope

Anything you need access to in the View should go on your $scope

Angular ‘watches’ everything on $scope

Don’t add anything to $scope unless the View needs

The $scope: contents

```
$scope.age = 12

$scope.pugs = [‘Pable’, ‘Rico’]

$scope.myPug = {age: 12, name: ‘Pablo’}

$scope.woof = function(pug){pug.bark()}
```

The $scope: Watches

Angular will watch $scope variables

When value changes in JS, it re-renders the relevant DOM

Your directives should fire the watch action when the model changes to reflect in JS

$watch whatever you want

You can set a callback function for anything that changes on the $scope

```
$scope.$watch('myPugs', function(){
   $window.alert('your pugs are changing!');
})
```

Deeper $watch

```
$watch(var, function) //works fine for vars

$watch(‘object’, f, true) //deep watch

$watchCollection(‘myPugsArray’, f)
//array contents
```

The $scope: $digest and $apply

`$digest`: calls watch on current $scope

`$apply`: calls $digest on **`$rootScope`** aka **`$digest`** on whole app

Slower machines/browsers will crap out at ~2000 concurrent watches

Creating App + controllers

```
//create app module, inject dependedcies
angular.module(‘appName’, [/*deps*/]);
//apps: lowerCamelCase, ctrl: UpperCamelCaseCtrl
angular.module(‘appName’) //gets module
   .controller(‘MyCtrl’,//create ctrl
[‘$scope’, function($scope){ //the ctrl function
   $scope.name = ‘grg’; //$scope enrichment
}]);
```

Dependency Injection

Every controller will depend on various components: from angular and written by you

Angular injects them based on name (eg will inject the **`$filter`** whenever your controller function has **`$filter`** as an arg)

Dependency Injection

JS minifiers will change your arg name, angular DI can no longer inject the correct one

Minification doesn’t touch strings, so we do:

```
[‘$scope’, ‘$filter’, function($scope, $filter){}]
```

Controller -> View

```
function MyCtrl($scope){
   $scope.pug = 'Pablo'
}
```

```
<div ng-controller='MyCtrl'>
   //can access anything on MyCtrl's $scope
   {{pug}} //Pablo
</div>
```

```
ParentCtrl = function($scope){
   $scope.note1 = ‘I Am A Parent’;
}
ChileCtrl = function($scope){
   $scope.note2 = ‘I am just a kid’;
}
<div ng-controller=‘ParentCtrl’>
   <div ng-controller=‘ChildCtrl’>
      {{note1}} //I Am A Parent
      {{note2}} //I am just a kid
   </div>
   {{note2}}//*nothing*
</div>
```

```
function ParentCtrl($scope){
  $scope.note = ‘I Am A Parent’;
}
function ChildCtrl($scope){
  $scope.note = ‘I am just a kid’;
}
```

```
<div ng-controller=‘ParentCtrl’>
  <div ng-controller=‘ChildCtrl’>
    {{note}} //I am just a kid
  </div>
  {{note}} //I Am A Parent
</div>
```

$scope functions

..can be called by many events

```
ng-focus ng-blur ng-change //inputs
ng-click ng-mouseover //elements
```

..or return values to

```
ng-repeat ng-class ng-if ng-show ng-hide
```