# INTRO TO ANGULAR 2 & COMPARISION WITH ANGULAR 1.X

NIKHIL KUMAR | SOFTWARE CONSULTANT

Knoldus Software LLP

# AGENDA

- INTRODUCTION to NG 2.0 & CONCEPTS
- WHY NEW VERSION ?
- MAJOR KEY FEATURES
- PROBLEMS IN 1.X & SOLUTIONS IN 2.0
- WEB COMPONENTS
- MIGRATION PATH TO ANGULAR 2

# HISTORY & STABILITY

AngularJS 1.3 is by far the best version of Angular available today. It was just released a few weeks ago. It's chock full of bug fixes, feature enhancements and performance improvements.

knólx

# Why Angular 2

A framework that cannot work with Web Components, bogs down on mobile or continues to push its own module and class API against the standards, is not going to last long. The Angular team's answer to these problems is a new version: Angular 2.0. It is essentially a re-imagining of AngularJS for the modern web, taking into account everything that has been learned over the last five years.


knólx

YEAHHHHH..... :D

# Web Components?

Web Components are on the horizon. The term Web Components usually refers to a collection of four related W3C specifications:

**Custom Elements** - Enables the extension of HTML through custom tags.
**HTML Imports** - Enables packaging of various resources (HTML, CSS, JS, etc.).
**Template Element** - Enables the inclusion of inert HTML in a document.
**Shadow DOM** - Enables encapsulation of DOM and CSS.

# ES ?

ECMA Script specification is a standardardized specification of a scripting language developed by Brendan of Netscape, initially it was named Mocha, later LiveScript and finally JavaScript.
In 1995 Sun & Netscape annnouced Javascript.

# Directives

In Angular 2.0 there will be three kinds of directives:

**Component Directives** – These will create reusable components by encapsulating logic in JavaScript, HTML or an optional CSS style sheet.

**Decorator Directives** – These directives will be used to decorate elements (for example adding a tooltip, or showing/hiding elements using `ng-show/ng-hide`).

**Template Directives** – These will turn HTML into a reusable template. The instantiating of the template and its insertion into the DOM can be fully controlled by the directive author. Examples include `ng-if` and `ng-repeat`.

# Structure Should/May be link:

```
root-app-folder
├── index.html
├── scripts
│   ├── controllers
│   │   └── main.js
│   │   └── ...
│   ├── directives
│   │   └── myDirective.js
│   │   └── ...
│   ├── filters
│   │   └── myFilter.js
│   │   └── ...
│   ├── services
│   │   └── myService.js
│   │   └── ...
│   ├── vendor
│   │   ├── angular.js
│   │   ├── angular.min.js
│   │   ├── es5-shim.min.js
│   │   └── json3.min.js
│   └── app.js
├── styles
│   └── ...
└── views
    ├── main.html
    └── ...
```

# Key Features

- **Mobile First**
- **Digest cycle**
- **Future Ready**
- **Speed & Performance**
- **Simple & Expressive**
- **Hierarchical Dependency Injection**
- **Support for Web Components**

# Differences with features

1- no digest cycle finished event

2- How Faster

3- Improved dependency injection

4- Directives Strategy etc

# No digest cycle finished event

$scope.$watch, $scope.$apply, $timeout.

```
$scope.$watch('variable',function(newValue,oldValue){
});
```

because such event might trigger further changes that kept the digest cycle going.

we had to reason about when to call `$scope.apply` or `$scope.digest`, which was not always straightforward

on occasion we had to call `$timeout` to let Angular finish its digest cycle and do some operation only when the DOM is stable

knólx

# No digest cycle finished event...

**Problems**:

Its not clear which watchers will be fired and in which order, or how many times
the order of the model updates is hard to reason about and anticipate
the **digest cycle can run multiple times(thats why no digest life cycle finished event)**
which is time consuming

**Solution**:

One of the first steps that the Angular team took in the direction of Angular 2,
was to extract from the Angular code base the mechanism of patching all asynchronous interaction points, and made it reusable.

# SOLUTION...

$scope.$watch, $scope.$apply, $timeout. No more. Whew! Using these was part of the reason Angular 1.x had such a huge learning curve.

**Zone.js** helps Angular to do change detection automatically. This sounds similar to React's reconciliation diffing algorithm.

```
element.addEventListener('keyup', function () {
  console.log('Key pressed.');
    });
});
```

# WHY ANGULAR 2.0 IS FASTER

1- Faster checking of a single binding

2- Avoid scanning parts of the component tree

# Improved dependency injection

Problem in Angluar 1.x

Angular 1 has a global pool of objects:

# Improved dependency injection

Solution in Angluar 2

In Angular 2 there will be only one DI mechanism:
constructor injection by type.

The fact that there is only one mechanism makes it easier to learn. Also the dependency injector is hierarchical, meaning that at different points of the component tree it's possible to have different implementations of the same type.

# Ng-directives

Components in the HTML are broken up into two types: (events) & [properties].

(events) refer to user initiated actions.

| 1.x | 2.0 |
|---|---|
| ng-click | (click) (dbl-click) |
| ng-keyup | (keyup) |

[properties] now link directly into the DOM properties.

| 1.x | 2.0 |
|---|---|
| ng-hide | [class:hidden] |
| ng-checked | [checked] |

knólx

***foreach**
!foreach is the proposed replacement for ng-repeat.

<ul> <li *foreach="#item in itemService.items"></li>  </ul>

**#item**
Items prefixed with a # can bind directly in the html. No more ng-model.

<input type="text" #userName />

# Migration Path to angular 2

The new Angular 2 router is being backported to Angular 1, and will allow the same application to have both Angular 1 and Angular 2 routes.

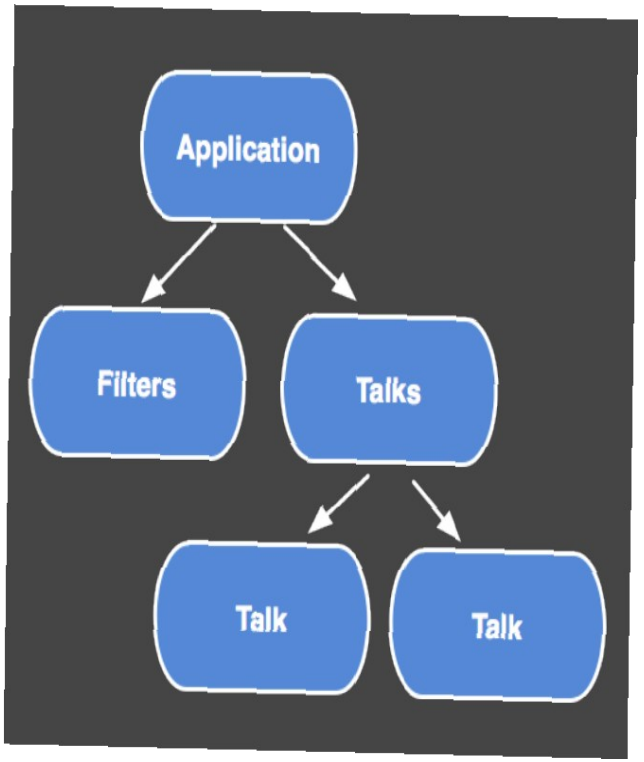
It will be possible to mix Angular 1 and Angular 2 components in the same application
It will be possible to inject services across framework versions.

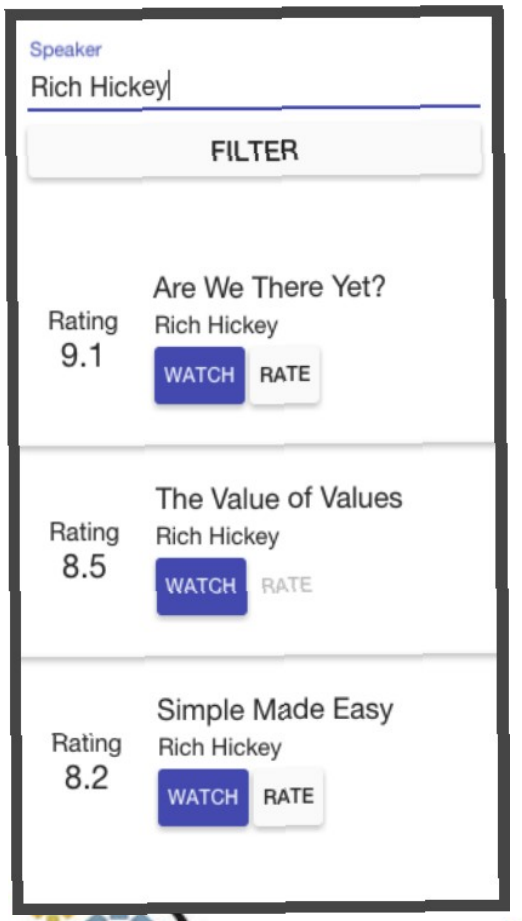
Data binding will work accross versions as the two change detection mechanisms will be integrated.

knólx

# Component...?

To build an Angular 2 application you define a set of components, for every UI element, screen, and route. An application will always have a root component that contains all other components. In other words, every Angular 2 application will have a component tree




knólx

# Component...

```
@Component({
  selector: 'talk-cmp',
  properties: ['talk'],
  events: ['rate']
})

@View({
  directives: [FormattedRating, WatchButton, RateButton],
  templateUrl: 'talk_cmp.html'
})
```

**Talk_cmp.html**

```
{{talk.title}}
{{talk.speaker}}
```

# Demo…

# References

http://tryangular2.github.io/#/section-1/page-1

http://www.codeproject.com/Tips/990533/Hello-Angular

http://eisenbergeffect.bluespire.com/all-about-angular-2-0/

# Thank You