Week 4

** Please only read this review AFTER watching* ***Lecture 34, Part 1*** *and* ***Part 2****.*

AngularJS – The **Publish-Subscribe** Design Pattern

The "Publish-subscribe design pattern" is implemented using the **AngularJS Event System**: you can publish events to (and listen for those from) anywhere in the application system.

- **TO PUBLISH**, you can use **$scope.$emit** to send an event **up** the scope chain or **$scope.$broadcast** to send an event **down** the scope chain. You can broadcast **from** the root scope to send an event to all nodes by using **$rootScope.$broadcast**.
- **TO LISTEN**, you can use either **$scope.$on** or **$rootScope.$on** listeners. Now, destroying the view destroys $scope.$on with it automatically because the listener is sitting in the same scope. In order to destroy $rootScope.$on, which is sitting in your entire application scope, you will need to deregister it manually by capturing the return value of **$on** method (the registration function) and calling it from inside the component's **$onDestroy()** method. Not doing it leaves the listener function in memory for the rest of the application's life, and every time you return to the same view new listener functions will be registered one on top of another, taking valuable memory resources.

Setting up a asynchronous action to communicate to the user to wait while something in your application is processing or loading (such as a spinner) **helps** the user understanding that their request is being processed (so they wont be clicking around wondering if the system froze or their last request was not computed for any reasons). In other words, showing some sign like a spinner tells the user something like "No further actions needed! Just wait a little bit and you will get your response! :)"

Consider, for example, an item is requested to be added to a **list**. You might run a bunch of asynchronous actions, and check whether the **item** is been already added to the list or should not be added to that list for any particular reason. Meanwhile, setting up a **spinner** to run on screen seems to be a good idea to let the user know they are going to get a response as soon as the **process** finishes.

1. **Show some indication** to the user that something is going on by displaying a ***loading spinner*** (an animated GIF) in a component **outside** the scope of our main controller (**ng-controller**='MainController as ctrl") but yet **inside** the scope of our Angular application (**ng-app**='MyApp').

```
<!DOCTYPE html>

<html ng-app='MyApp'>

  <head>

    // tags for utf-8, css file and title go here

  </head>

  <body>

    <h1>My App Title</h1>

    <div ng-controller='MainController as ctrl'>

      (...)

      <main-component

        // main component properties

      >

      </main-component>

    </div>

    <loading-spinner-component></loading-spinner-component>

    // script tags for our .js files:

    <script src=”jquery.min.js”></script>

    <script src=”angular.min.js”></script>

    <script src=”app.js”></script>

  </body>

</html>
```

2. Create our **spinner**.html **template** to show/hide our "loading" spinner:

```
<img ng-if=$ctrl.showSpinner"

  class="loading-icon"

  scr="loading.gif"

  alt="loading">
```

3. Define a "**loadingSpinnerComponent**" function inside our **app.js** application and configure its **controller** for listening and to show/hide the spinner according to a message being broadcasted. Also, configure to start/stop broadcasting the message in "**MainComponentCtrl**" function:

```
(...)
```

```
angular.module('MyApp', [])

  .controller('MainController', MainController)

  .service('AsyncFilterService', AsyncFilterService)

  .component('mainComponent', {...controller: MainComponentCtrl...})

  .component('loadingSpinnerComponent', {

    templateUrl: 'spinner.html',

    controller: SpinnerController

  });

SpinnerController.$inject = ['$rootScope']

function SpinnerController($rootScope) {

  var $ctrl = this;

  var destroy = $rootScope.$on('application:processing', function

      (even, data) {

    $ctrl.showSpinner = data.on;

  });

  $ctrl.$onDestroy = function () {

    destroy();

  };

};

MainComponentCtrl.$inject = ['$rootScope', '$element', '$q',

    'FilterService']

function MainComponentCtrl($rootScope, $element, $q, FilterService)

    {

  var $ctrl = this;

  var total;

  $ctrl.$onInit = function () { total = 0; };

  $ctrl.$doCheck = function () {

    if ($ctrl.items.length !== total) {

      total = $ctrl.items.length;

      $rootScope.$broadcast('application:processing', {on: true});
      var promises = [];

      for (var i = 0; i < $ctrl.items.length; i++) {

        promises.push(AsyncFilterService.checkName($ctrl.items[i]

            .name));

      }

      $q.all(promises).then(function (result) {

        $element.find('div.error').slideUp(1000);

      }).catch(function (result) {

        $element.find('div.error').slideDown(1000);

      }).finally(function () {

        $rootScope.$broadcast('application:processing', {on: false}

            );

      });

    }

  };

};

(...)
```