Week 4

coursera.org/learn/single-page-web-apps-with-angularjs/discussions/weeks/4/threads/knFvnhDXEee0Lw6LyVitXg

** This is a review from the all lectures of **Week 4**'s **"Lesson 1 - Components and Component-Based Architecture"** (**Lecture 33, Parts 1/2/3/4**).*

AngularJS – The **Component**-based Architecture

Component-based architecture views the application as a **Tree of Components** where the entire application is comprised of components, being each with well-defined input and output, and where two-way data binding should be minimized as much as possible.

COMPONENT's **Key Features**

- Have **ISOLATE SCOPE** with well-define API;
- **API** was added into **AngularJS 1.5** to improve your application and to update you for **AngularJS 2** which in turn is mostly exclusively based on components;
- Is registered with **name** and configuration **object**;
- Simplifies the **directive**'s configuration by assuming defaults;
- Have a **built-in controller** although you may set your own for extra functionality you might want to implement;
- DOES NOT restrict you to component-based architecture but **makes it easier to structure** you application code.

a) Component's **Isolate Scope**

Because of **prototypal inheritance**, it is possible for your code to modify data pretty much anywhere in your application. To avoid side-effects that lead to chaos, Angular components **always** use **isolate scope**, which means they only control their **own** View and Data – they never modify data or DOM outside their own **scope**.

b) Component's **Well-defined public API** (**Inputs & Outputs**)

Component-based architecture follows conventions design to discourage manipulation of data which does not belong to the component:

- **Input**: only use '**<**' and '**@**' bindings (never the '=' bidirectional binding);
- **Output**: only use '**&**' reference binding, passing data to callback with {key:value} param map (never change the property value of a passed object or array)

c) Component's **Lifecycle Pre-defined Methods**

- **$onInit**() – controller initialization code;
- **$onChanges(**changeObj**)** – one-way bindings update;
- **$postLink**() – similar do "link" in directive but without the parameters;

- **$onDestroy**() – when scope is unloaded from memory;
- **$doCheck**() – called on each turn of the digest cycle (added in v. **1.5.8**);
- Check them all and more on https://docs.angularjs.org/guide/component

d) Component's **Creating Steps**

1. Set Component in the Parent HTML:

```
<my-component

  prop1="{{parentProperty1}}"

  prop2="parentProperty2"

  on-action="$ctrl.myParentActionMethod(myArg)">

</my-component>
```

2. Register Component: provide a **name** ('myController') and a **configuration object** straight in line (instead of having a function for defining the component).

3. Configure Component: Inside the **object**: a controller is not required unless you want to specify some functionality to it (then you set something like **controller: MyComponentController**). If you don't, Angular will provide an empty controller object automatically. In any case, Angular will always provide the '**$ctrl**' label to one controller or the other; The traditional directive's scope is now called "**bindings**" because in a component the scope is always assumed isolated.

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

  .component('myComponent', {

    templateUrl: 'template.html',

    controller: MyComponentController,

    bindings: {

      prop1: '@',

      prop2: '<',

      onAction: '&'

    }

  }

});
```

4. Set Reference Properties in Template (the ones passed into the component): the **$ctrl** label in the example below is automatically set by Angular.

```
<h3>{{ $ctrl.prop1 }}</h3>

<ol>

  <li ng-repeat="item in $ctrl.prop2">

    {{ item.quantity }} of {{ item.name }}

    <button ng-click="$ctrl.onAction({ myArg: $index })">My Action

        </button>

  </li>

</ol>

<div class="error" ng-if="$ctrl.myConditionalMethod()">My Message

    </div>
```

5. Configure the optional controller: Declare **MyComponentController** which is also the place you would use the component's **lifecycle pre-defined methods**.

```
MyComponentController.$inject = ['searchString', '$element'];

function MyComponentController(searchString, $element) {

  var $ctrl = this;

  var total;

  $ctrl.myConditionalMethod = function(searchString) {

    for (var i = 0; i < $ctrl.prop2.length; i++) {

      var name = $ctrl.prop2[i].name;

      if (name.toLowerCase().indexOf(searchString) !== -1) {

        return true;

      }

    }

    return false;

  };

  $ctrl.$onInit = function () {

    console.log("Inside $onInit()");

    total = 0;

  };

  $ctrl.$onChanges = function (changeObj) {
```

```
    console.log("Changes: ", changeObj);
  };
  $ctrl.$doCheck = function () {
    if ($ctrl.prop2.length !== total) {
      total = $ctrl.prop2.length;
      var warningElem = $element.find('div.error');
      if ($ctrl.myConditionalMethod()) {
        warningElem.slideDown(1000);
      } else {
        warningElem.slideUp(1000);
      }
    }
  };
  $ctrl.$postLink = function () {
  };
  $ctrl.$onDestroy = function () {
  };
}
```

6. Since we are using jQuery features in $doCheck() above, do not forget to reference it in the main html page (index.html) before angular.min.js (or angular.js):

```
<script src=”jquery.min.js”></script>
<script src=”angular.min.js”></script>
<script src=”app.js”></script>
```