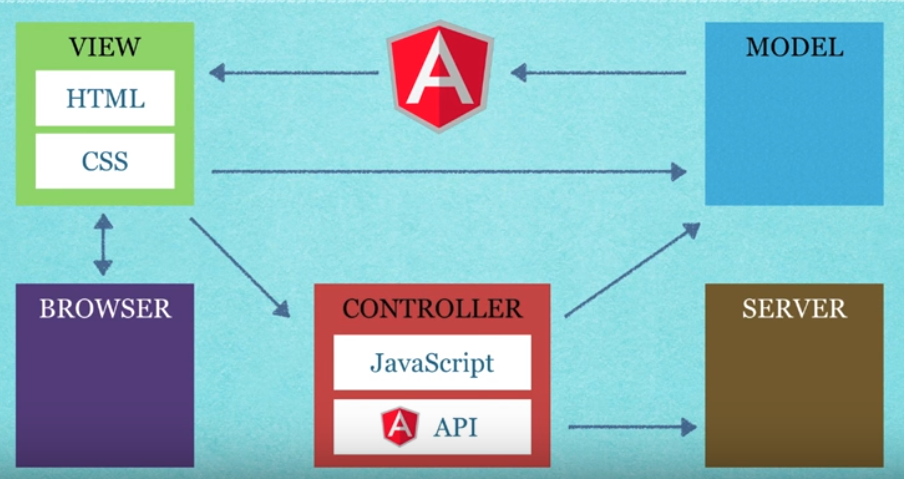
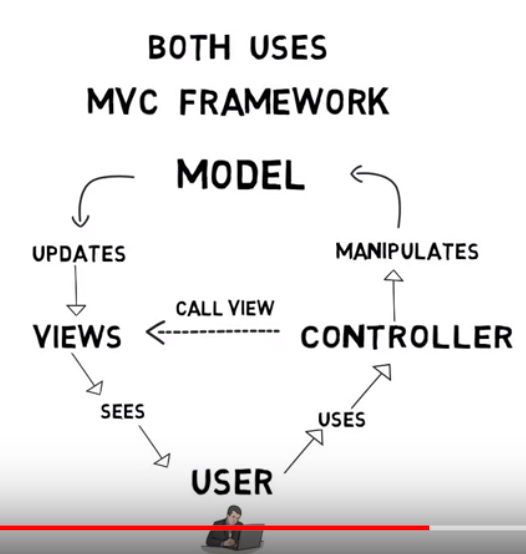
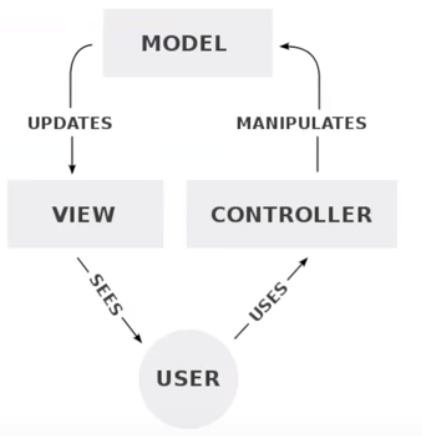
Angular JS (Front End)

Angular and **jQuery** can't reasonably be compared. Angular is a framework, **jQuery** is a library. Frameworks have their place and libraries have their place. However, there is no question that a good framework has more power in writing an application **than** a library.

It's not a **good** idea to **use jQuery with angular**.js. **jQuery** does different thing and**angular**.js does different things. **jQuery** manipulates the DOM and **add**, append, remove element dynamically. In other hands, **angular**.js just manipulate model data but you can **use** jqlite built in with **angular**.js.

\*Other Stuff\*

* Backbone Framework – uses model view controller in html to accomplish what JS, jQuery does

 **MVC:** 

**Front End**:

* Manipulates DOM
* Created responsive, single page applications
* Provides a frontend framework based on the MVC model

**YouTube Tutorial**: <https://www.youtube.com/watch?v=OPxeCiy0RdY>

Angular JS Jargon:

* Modules: represent the components used in your application
* Using modules makes it much easier to reuse these components in other applications or other parts of your site
* \*Used to associate an Angular app with the part of the HTML document

\*Web pages are normally manipulated by working with the DOM object, w/ JavaScript & jQuery. Angular is going to allow you to extend HTML attributes using directives\*

* Directives: make it easy to bind data directly to your HTML elements

\*Angular is going to use JavaScript objects to represent said data (called scope)

* Scope: links our HTML elements to variables in our scope
* data – can be data generated on the server, or database, or web service, or client side using Angular.js code
* Expressions: directly linked to the scope (data) – benefit 🡪 page will be updated dynamically as the data changes

\*Data Binding: works so

* Whenever data change on the web page – the model will be updated
* Whenever data changes on the model – the web page will be updated
* Services:
* AJAX techniques
* Controllers – can access the angular model in JS via the controller
* $Scope Object:
* is the model in Angular
* links our HTML elements to variables in our scope

Angular.js Tutorial Web Page:

* ng-app: tells Angular where to start compiling the code (usually on html or body element)
* ng-init: initializes application data by assigning different values to variables (name value pairs)
* ng-controller: defines the element / portion of the page as a view
* use scope component/object to provide data to the view
* ng-model: binds the value of HTML elements to values defined in the scope (application data)
* ng-bind: binds application data to the HTML view
* ng-repeat: can be used to cycle through a list of data – very similar to a for each loop

\*Create multiple views that use the same one controller

\*Create multiple controllers

\*When refreshing the page, you can see curly brackets for a brief second\*

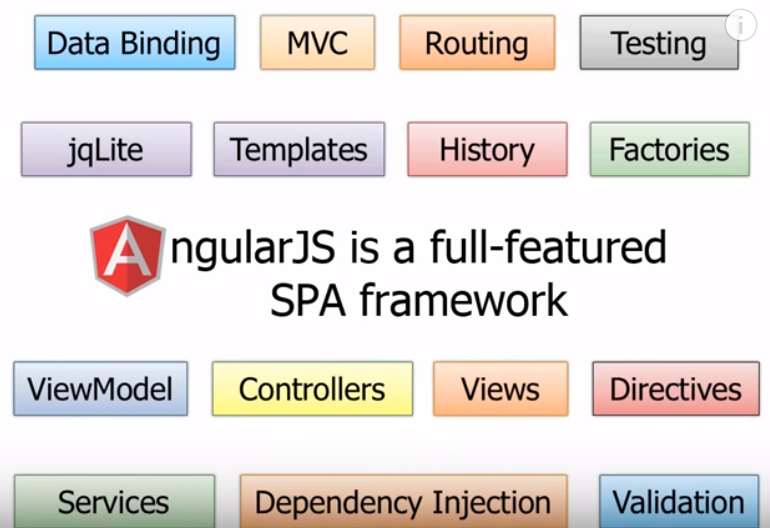
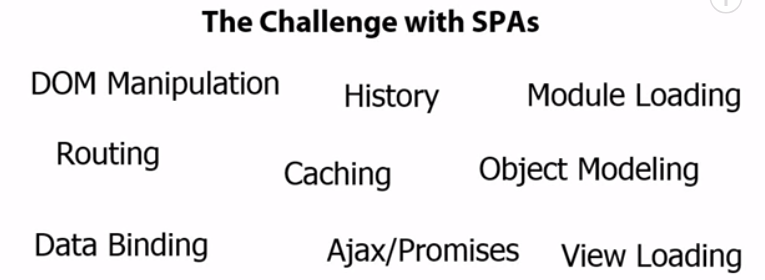
* Angular ng-cloak directive & CSS cloak style
* Put your Angular reference at the top of the page instead of at the end of the body section
* These –

\*Can bind data using the directive ‘ng-bind’ or by using expressions\*

**Various Terms**:

* Factory Function
* Dependency: parameter passed in the factory function (in the ‘attributes area’)
* Dependency Injection
* Unary Plus Operator -

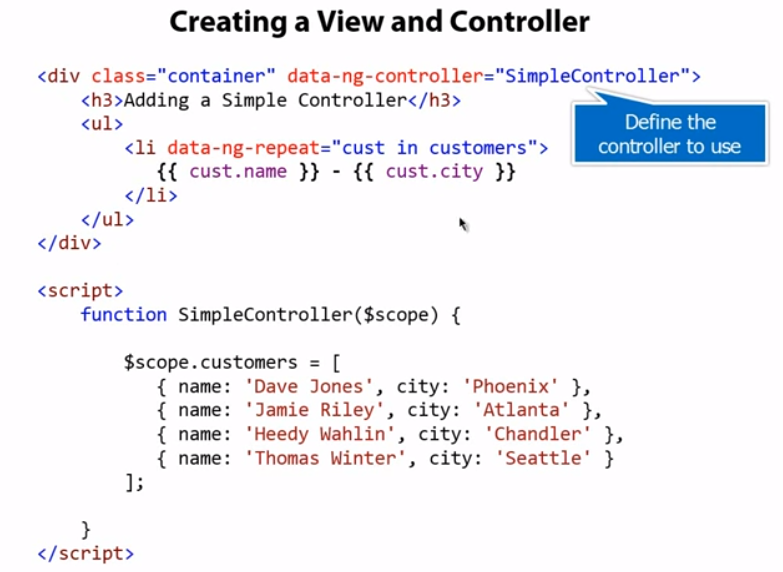
[**https://www.youtube.com/watch?v=i9MHigUZKEM**](https://www.youtube.com/watch?v=i9MHigUZKEM)



**Directives** –

* ng- represents a built in Angular directive; can also write your own, use 3rd party
* Teaches HTML new tricks
* Extends HTML easily
* \*Add ‘data-‘ before directives to take advantage of validation features? (look into more)
* **ng-app**= “app1” 🡪 initializes the Angular app
* gives the page the go ahead to start using angular features
* can represent an association with a module called “app1”
* **ng-model**= “name” 🡪 adds a property in a memory region called $scope
* in essence it is creating an empty ViewModel behind the scenes & adding a property called “name” to it
* the property’s value will equate the ….
* Value user enters into the html control?
* Value initialized in the JS within the $scope object - $scope.name = “”?
* \*To write out / display that value – can add a Data Binding Expression 🡪 {{name}}
* Expression

Filters –



**Views, Controllers, Scope (ViewModel)**: (MVC)

**Views** –

* What we’ve been working with by using directives, data binding, & filters
* BUT it’s not ideal to put all of our logic in the view
* not very maintainable or testable
* instead we use a Controller

**Controller** –

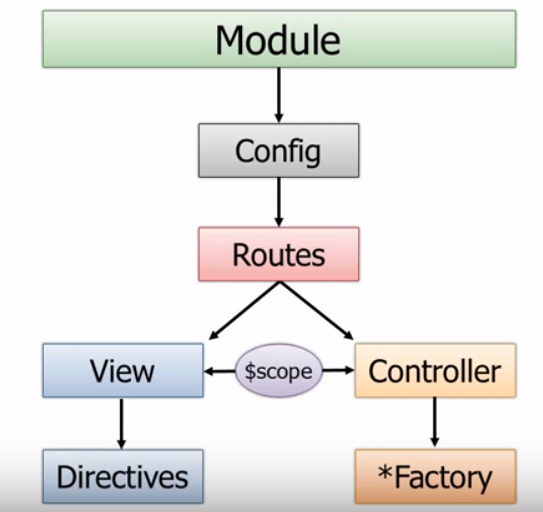
* Drives things – controls what data gets bound into the view
* The controller itself is not called by the view – the $scope is
* The $scope is implicitly available to the portion of code in the HTML that defines the corresponding controller
* View might pass data to controller that the controller then will pass to a data service

\*The glue between the View & the Controller is **$scope** object\*

**$scope** –

* The ViewModel – the Model aka data for the View
* Ties the Controller to the view
* Neither needs to or wants to know about each other
* The view can know about the controller using a specific directive
* The controller should not know anything about the view to keep things loosely coupled
* Controller should be able to bind with multiple views: mobile, desktop, etc.
* Provides the built-in dependency injection of Angular
* The $scope object is passed as a parameter into the controller function
* Within the function, we can take that object & add properties, methods onto it
* With this info, the controller can then serve as the source of the data for the view
* The $scope will be automatically bound into the view once the view knows about the controller (this way the view & controller don’t need to directly interact)

Modularity (Modules), SPA oriented concepts: Routes, Factories –

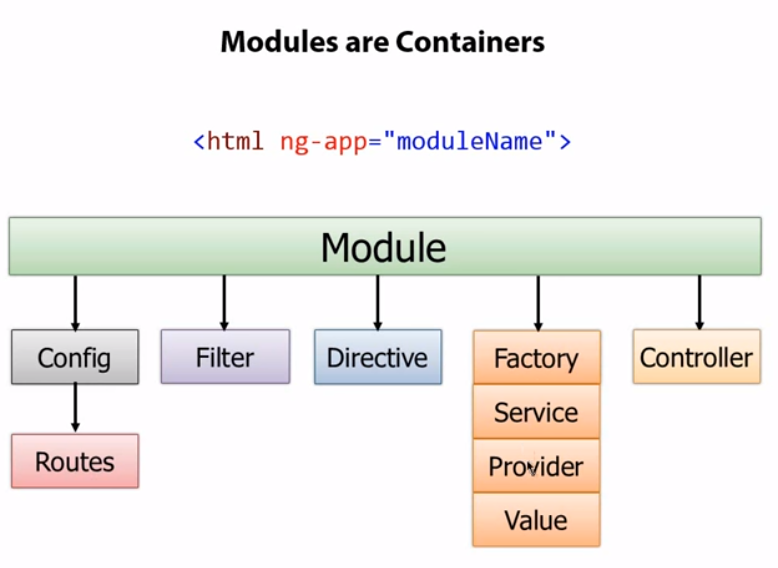
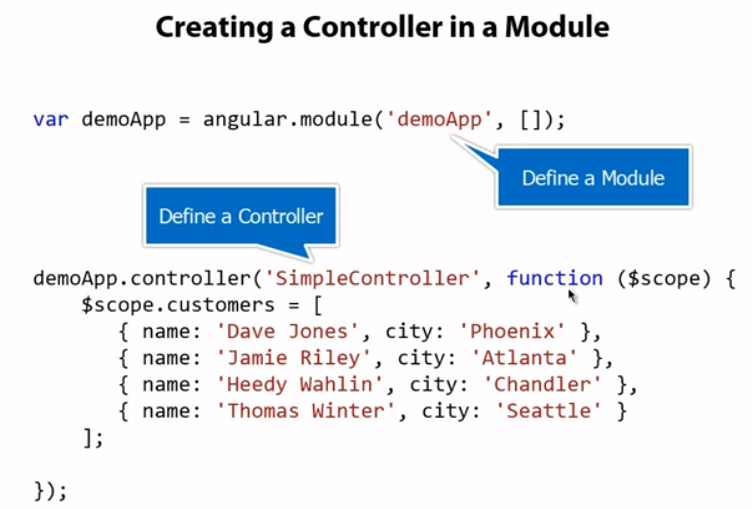
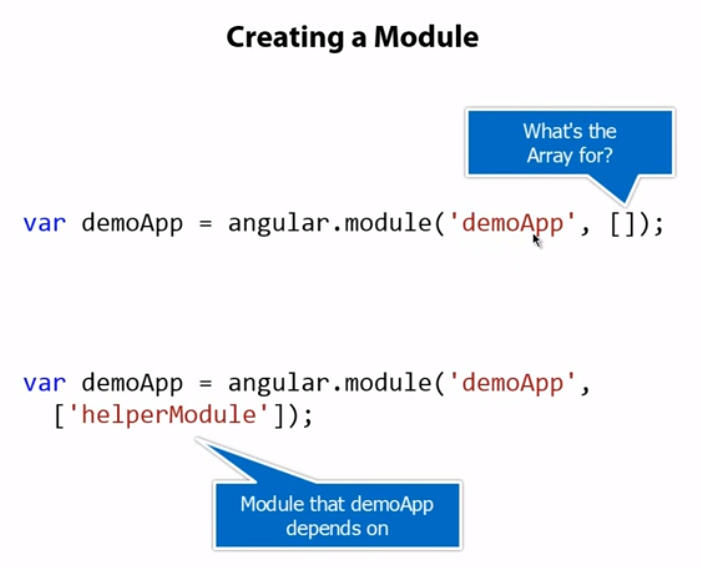


Modules –

* Module can have a Config Function to define different Routes
* When you define a Route – can define a
* View
* Directives, Filters
* Controller
* Can call out to Factories, Services, Providers, Values, Resources to access data

Angular’s Module Object – angular.module (Object Container)

* Off the Module you can:
* Configure the routes
* Create custom Filters & Directives
* Get data from different sources
* Create Controllers
* Once the module is given a name – must specify it on the ng-app directive

Why routes important in SPA world?

* When you have different Views, those views need to be loaded into the shell page, need to track what route we are on, what view & controller is associated with that route

**Donnie’s Class Notes:**

**OVERVIEW:**

* Extends HTML w/ new attributes
* Perfect for Single Page Applications (SPAs)
* Basics of Angular:
* Directives, Expressions, Modules, Controllers, Filters
* Other Areas:
* Events, DOM, Forms, Input, Validation, HTTP

**INTRODUCTION:**

\*Angular 🡪 a JavaScript framework – can be added to an HTML page w/ a <script> tag

\*Angular extends HTML attributes w/ Directives & binds data to HTML w/ Expressions

* AngularJS is a JS framework 🡪 a library written in JavaScript
* Starts automatically when the web page has loaded
* Is distributed as a JavaScript file & can be added w/ a script tag:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>

**Directives:**

\*HTML attributes w/ an ng prefix

* **ng-app** directive:
* defines the application
* **creates a scope for an angular application**
* tells AngularJS that the outer most <div> element is the "owner" of an AngularJS **application**
* **ng-model** directive:
* marks the HTML element as a data model
* something else in app will use the value of this model
* binds the value of the input field to the application variable **name**
* **ng-bind** directive: (alternative to expressions)
* takes the value of the model and puts it in the given element
* binds the innerHTML of this element to the value in the corresponding model
* binds the **innerHTML** of the given element to the application variable **name**

**Expressions:**

* AngularJS expressions are written inside double braces 🡪 **{{ expression }}**
* will "output" data exactly where the expression is written
* Expressions bind AngularJS data to HTML the same way as the **ng-bind** directive

**AngularJS Applications:**

* \*Modules 🡪 define
* \*Controllers 🡪 define
* ng-app directive 🡪 defines the application
* ng-controller directive 🡪 defines the controller

**EXPRESSIONS:**

* AngularJS expressions can be written inside double braces 🡪 {{ *expression* }}
* can also be written inside a directive 🡪 ng-bind="*expression*"
* AngularJS will resolve the expression & return the result exactly where the expression is written
* **AngularJS expressions** are much like **JavaScript expressions:** They can contain literals, operators, and variables 🡪 Ex: {{ 5 + 5 }} or {{ firstName + " " + lastName }}

\*Let Angular change the value of CSS properties\*

\*AngularJS 🡪 Number, Strings, Objects, Arrays

* AngularJS Expression VS JavaScript Expressions
* Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.
* AngularJS expressions can be written inside HTML – JavaScript Expressions cannot
* AngularJS expressions support filters – JavaScript expressions do not.
* AngularJS expressions do not support conditionals, loops, and exceptions – JavaScript expressions do.

**MODULES:**

\*AngularJS module defines an application / portion of an application

* Is a container for the different parts of an application
* Is a container for the application controllers
* Controllers always belong to a module

**DIRECTIVES:**

* Data Binding:
* Repeating HTML Elements:
* **ng-repeat** directive: repeats an HTML element
* **clones HTML elements** once for each item in a collection
* **ex: use on an array of objects**

**CONTROLLERS:**

\*Control the data of AngularJS applications

\*Is a JS Object – created by a standard JS Object Constructor

\*ng-controller 🡪 defines the application controller

<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>

* The AngularJS application is defined by **ng-app="myApp"**. The application runs inside the <div>
* The **ng-controller="myCtrl"** attribute is an AngularJS directive. It defines a controller
* The **myCtrl** function is a JavaScript function
* AngularJS will invoke the controller with a **$scope** object
* In AngularJS, $scope is the application object (the owner of application variables and functions)
* The controller creates two properties (variables) in the scope (**firstName** and **lastName**).
* The **ng-model** directives bind the input fields to the controller properties (firstName and lastName)