CSC 443: Web Programming

ANGULARJS

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

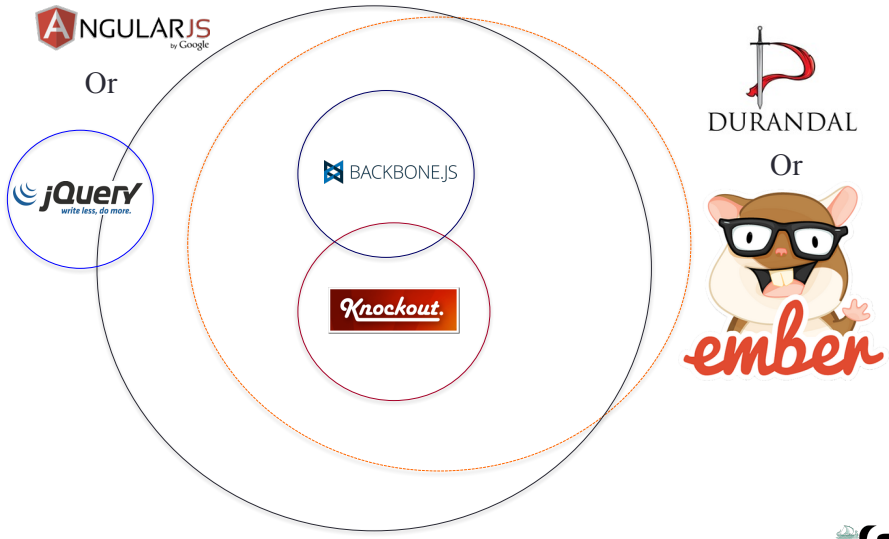
What is ANGULARJS ?

- ANGULARJS is a framework (a JavaScript library) that makes it easier to communicate between your HTML document and JavaScript.

Other Frameworks

Marionette
BACKBONE.JS
ember
Meteor JS
dojo
canjs
Knockout.
React

Angular.js vs other libraries



vs jQuery

- jQuery is a library meant for DOM manipulation, animations and an AJAX wrapper
  - NOT an application framework (in the strict sense)
- Pros
  - Angular has built in 'basic' jQuery.
    - If full-blown jQuery is added Angular will automatically use it. Generally full blown NOT needed.
- Cons
  - Not a good choice for creating dynamic UIs.
    - Verbose code, hard to maintain, not organized
    - Not MVVM or MVC

vs BACKBONE.JS

- Provides structure to web applications as well as model, view, templating, basic routing and RESTful interactions.
- Pros
  - Older and more mature, more people using it
- Cons
  - "Previous generation" web app framework
    - No MVVM w/o add-ons – use jQuery for DOM manipulation
    - No DI, not as easily testable
    - Not a full-featured framework, meant to be 'light'
    - Extremely verbose for what you get
- Not actively developed

vs Knockout.

- A library that provides MVVM data bindings using observables and dependency tracking
- Pros
  - Possibly more performant under some situations
- Cons
  - Complicated and error prone
    - Dependency tracking, computed's get confusing
    - "when complicated things break it's hard to figure out why"
    - No POJO. Have to create "types" and ko.observable()s
- All it does is MVVM, not an app framework
  - Testability, code organization etc. all potential issues

vs ember

- A full-fledged framework for web applications
- Pros
  - Similar goals as Angular.js
- Cons
  - Uses observables, special objects, string getters and setters, not dynamic
  - Is ideal for LARGE web apps
  - Is arguably more complicated
  - Very opinionated, have to use their object "bases"
  - Not as popular as Angular

On to ANGULARJS by Google

Features of AngularJS

- Two-way Data Binding
  - Model as single source of truth
- Directives
  - Extend HTML
- MVC
- Dependency Injection
- Testing
- Deep Linking (Map URL to route Definition)
- Server-Side Communication

Starting with AngularJS

- Download AngularJS from https://angularjs.org

Illustrating ANGULARJS Using a Simple Example

- Create a form that allows a user to enter his/her name
  - The application greets the user while entering the name.
  - Once you click submit, the application uses AJAX to send the name to a web service.

Iteration 1: Create the form

```
<form>
   <p>
      Enter your name:
      <input type="text" id="name" required>
      <button type="button"
              disabled="disabled”>Submit</button>
   </p>
   <p id="greet" style="display: none">
      Hello,
      <span id="name-repeat"></span>
   </p>
</form>
```

Iteration 1: jQuery Implementation

```
var name = $('#name'),
    nameRepeat = $('#name-repeat'),
    greet = $("#greet"),
    submit = $("button");

name.bind('keyup', function() {
    var disabled, value;
    value = name.val();
    if (value.length === 0) {
        submit.attr('disabled', 'disabled');
        greet.css('display', 'none');
    } else {
        submit.removeAttr('disabled');
        greet.css('display', 'block');
    }
    nameRepeat.innerHTML = value;
});

submit.bind('click', function() {
    $.ajax('/service?name=' + name.val());
});
```

Iteration 2: AngularJS

```
<form>
   <p>
      Enter your name:
      <input type="text" ng-model="name" required>
      <button type="button" ng-click="submit()"
              ng-disabled="!name">Submit</button>
   </p>
   <p ng-show="name">
      Hello, {{name}}
   </p>
</form>
```

```
$scope.submit = function() {
    $http.get('/service?name=' + $scope.name);
};
```

The $scope variable links the controllers and view
ng-show directive shows or hides an HTML element
ng-click directive allows to specify a custom behavior when an element is clicked.
ng-disabled sets the disabled attribute on the element if the expression inside ngDisabled evaluates to true

Example 2: A Complete AngularJS Application

```
<div ng-app=""> or <div ng-app>
<p>Input something in the input box:</p>
<p>Name : <input type="text" ng-model="name"
placeholder="Enter name here"></p>

<h1>Hello {{name}}</h1>

</div>
```

The critical Foundation for understanding

TERMINOLOGY

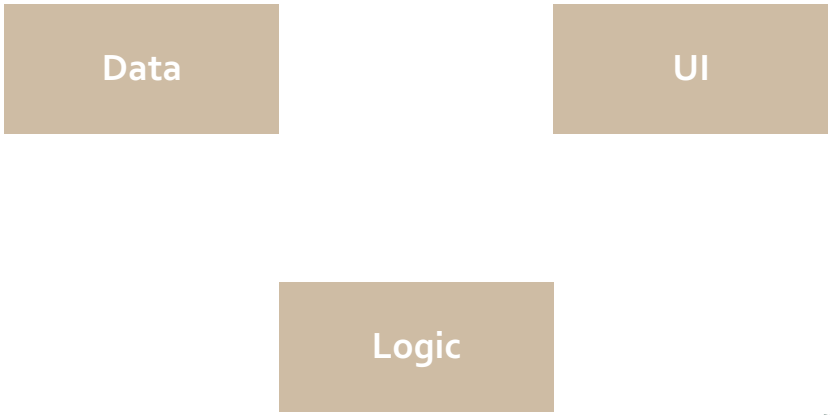
Model View Controller

- What is the MVC?
  - Model – the data
  - View – the user interface, what the user sees and interact with
  - Controller – the interface between the model and the view

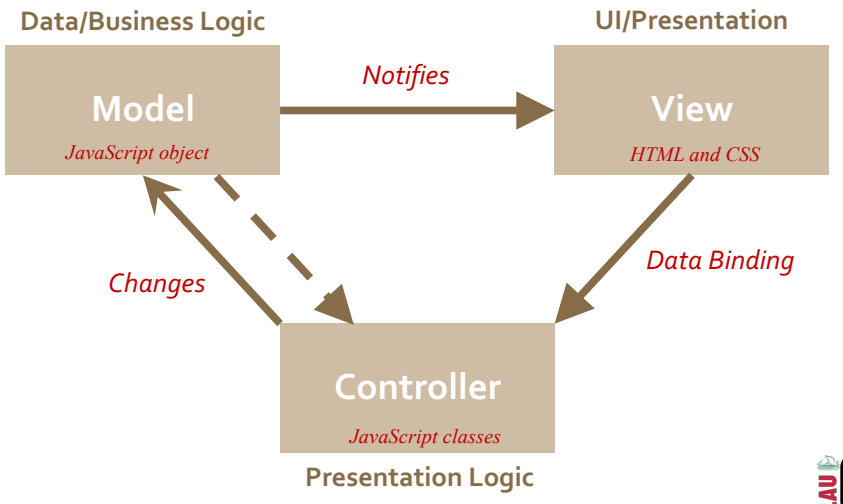
Model View Controller

- What is the MVC?
  - The model is not necessarily the data from the database.
  - The browser, server, and database can have their own MVC systems and often do.
  - When we talk about the MVC in this lecture we'll be talking about the MVC on the browser.

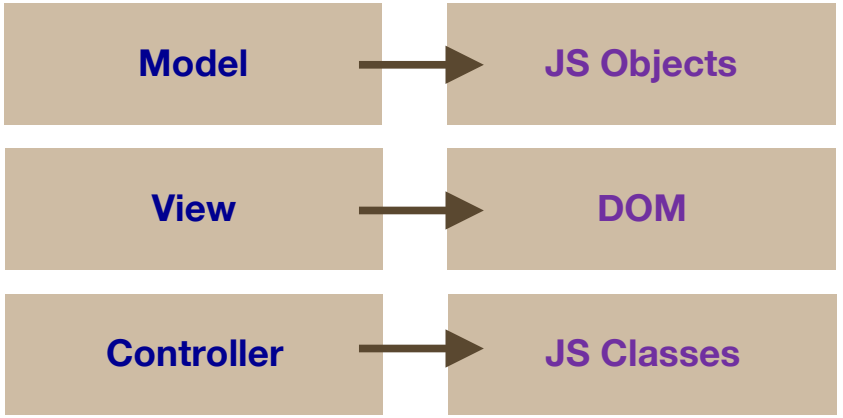
Structure

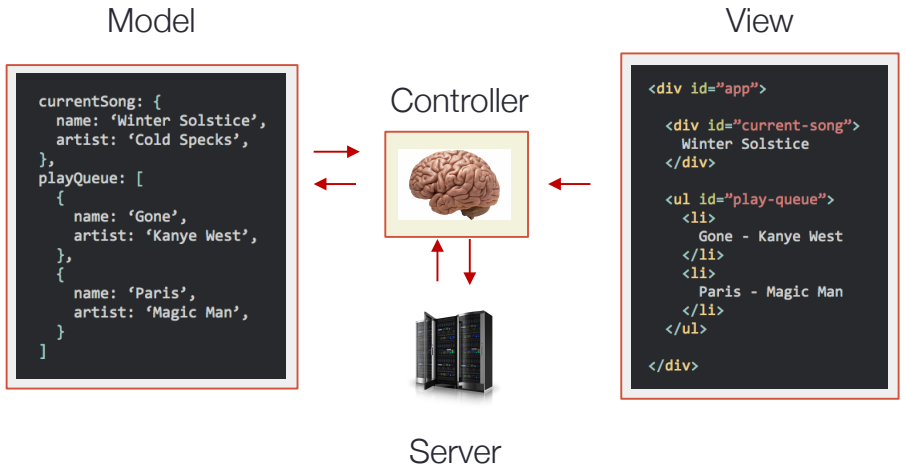


Structure: MVC



MVC





A First Angular Example

- A client-side Angular templating
  - ng-controller provides an identifier to link with code
  - ng-repeat iterates through a variable in the controller's $scope

```
<h1>Starter AngularJS app</h1>
<div ng-controller="ToddlerCtrl">
  <h2>Toddlers</h2>
  <table>
    <tr>
      <th>Name</th>
      <th>Birthday</th>
      <th>Happy?</th>
    </tr>
    <tr ng-repeat="toddler in toddlers">
      <td>{{toddler.name}}</td>
      <td>{{toddler.birthday}}</td>
      <td>{{toddler.happy}}</td>
    </tr>
  </table>
</div>
```

Controller

- The name of the app is myApp
- Controller is ToddlerCtrl
  - Define the controller and fill our scope up with Toddler objects.
    - Just a Javascript data structure - JSON

```
// Instantiate the app, the 'myApp' parameter must
// match what is in ng-app
var myApp = angular.module('myApp', []);

// Create the controller, the 'ToddlerCtrl' parameter
// must match an ng-controller directive
myApp.controller('ToddlerCtrl', function ($scope) {

  // Define an array of Toddler objects
  $scope.toddlers = [
    {
      "name": "Toddler One",
      "birthday": "1/1/2011",
      "happy": true
    },
    {
      "name": "Toddler Two",
      "birthday": "2/2/2011",
      "happy": true
    },
    {
      "name": "Toddler Three",
      "birthday": "3/3/2011",
      "happy": false
    }
  ];

});
```


$scope

- $scope is the glue between the controller and the view.
- More about this later!

Using client-side models from different data sources

- Data sources
  - Using JSON in the initial page load to pre-populate Services
  - Using a static JSON file
  - Using a REST API
- We'll build on the previous example by creating our Models using different data sources.

Using JSON in initial page load

- Assign the JSON to a variable in a <script> tag.
- Create a Service using $resource
- Pass the JSON to the constructor

```
<script type="text/javascript">
// Define an array of Toddler objects
window.toddlers = [
  {
    "name": "Toddler One",
    "birthday": "1/1/2011",
    "happy": true
  },
  {
    "name": "Toddler Two",
    "birthday": "2/2/2011",
    "happy": true
  },
  {
    "name": "Toddler Three",
    "birthday": "3/3/2011",
    "happy": false
  }
];
</script>
```

```
myApp.factory('Toddler', function($resource) {
  return $resource('toddlers.json');
});

$scope.toddlers = [];
angular.forEach(window.toddlers, function (item) {
  $scope.toddlers.push(new Toddler(item));
});
```

Using REST API & JSON file

- Create a URL template.
  - This identifies the object ID field (aid)
- With the same Adult resource, you do a get() to request a single object


```
/**
 * Adult is a service that calls a REST API
 * It's not really a REST API, but just calling our local .json file
 * as an example
 * If you call Adult.query(), it will GET adults.json
 * If you call Adult.get({}, {aid: 1}) it will GET adults/1.json
 */
myApp.factory('Adult', function($resource) {
  return $resource('adults/:adultId.json', {adultId: '@aid'});
});

// Create the controller, the 'PersonCtrl' parameter must
// match an ng-controller directive
myApp.controller('PersonCtrl', function ($scope, Toddler, Teen, Adult) {

  // Initialze Toddlers from JSON defined on initial page load
  $scope.toddlers = [];
  angular.forEach(window.toddlers, function (item) {
    $scope.toddlers.push(new Toddler(item));
  });

  // Teens are from static json file
  $scope.teens = Teen.query();

  // Adults are from REST API
  $scope.adults = Adult.query();

  // Example of grabbing single Adult from REST API
  $scope.singleAdult = Adult.get({}, {aid: 1});

});
```

Templating

- Directives + AngularJS templating allows you to create custom HTML markup, both elements and attributes
- Templating types:
  - We've already seen inline HTML
  - Can define within Javascript
  - Can include within <script> tags
  - Can include in an external HTML file
- We'll take our existing code, pick the local JSON file as the data source, and show a comparison between these different templating methods.

Templating: JavaScript

```
/**
 * Use a javascript-based template
 */
myApp.directive('teenJavascript', function() {
  return {
    restrict: 'AE',
    scope: {
      teen: '='
    },
    template: '<tr><td>{{teen.name}}</td><td>{{teen.birthday}}</td><td>{{teen.happy}}</td></tr>'
  };
});
```

- Use an AngularJS directive
- Can also declare a template right in Javascript

Templating: Script

```
<script type="text/ng-template" charset="utf-8" id="teen-internal.html">
  <tr>
    <td>{{teen.name}}</td>
    <td>{{teen.birthday}}</td>
    <td>{{teen.happy}}</td>
  </tr>
</script>
```

- Template cache can be pre-loaded by including a template within <script> tags, using a special type.
- Any directive that references teen-internal.html first looks in template cache before requesting the html file from the server.

Back to Basics…

Two-Way Data Binding

- You can bind a variable in `$scope` to elements or inputs
- You can also use a `$scope` variable to control which CSS class gets applied to an element
- If the input changes its value, the underlying $scope variable also changes.
- To achieve 2-way data binding, you use `ng-model` to attach an element to something in the `$scope`
- The ng-class construct also applies the appropriate class as the underlying model changes
- You can also do 1-way data binding using `ng-bind` instead of `ng-model`

Two-Way Data Binding

- Data-binding in Angular apps is the automatic synchronization of data between the model and view components. (https://docs.angularjs.org/guide/databinding)
- From the previous AngularJS example we have several data binding instances (marked below in red).

```
<form>
   <p>
      Enter your name:
      <input type="text" ng-model="name" required>
      <button type="button" ng-click="submit()" ng-disabled="!name">Submit
</button>
   </p>
   <p ng-show="name">
      Hello, {{name}}
   </p>
</form>
```

Data Binding

```
<html ng-app>
<head>
  <script src='angular.js'></script>
</head>
<body ng-init="user.name = 'Larry'">
  <div>Hi {{user.name}}</div>
</body>
</html>
```

Expressions

- A JavaScript expression produces a value and can be written wherever a value is expected. (http://www.2ality.com/2012/09/expressions-vs-statements.html)

```
myvar
3 + x
myfunc("a", "b")
```

Expressions

- AngularJS also has expressions:
  - Angular expressions are JavaScript-like code snippets that are usually placed in bindings such as {{ expression }}. (https://docs.angularjs.org/guide/expression)

```
<form>
   <p>
      Enter your name:
      <input type="text" ng-model="name" required>
      <button type="button" ng-click="submit()" ng-disabled="!name">Submit
</button>
   </p>
   <p ng-show="name">
      Hello, {{name}}
   </p>
</form>
```

More Angular Expressions

- Arrays
  - ng-init="points=[1,15,19,2,40]"
- Objects
  - ng-init="person={firstName:'John',lastName:'Doe'}"
- Strings
  - ng-init="firstName='John';lastName='Doe'"
- Numbers
  - ng-init="quantity=1;cost=5" or simply {{ 5 + 5 }}
- CSS Properties
  - ng-init="myCol='lightblue'"

Directives

- A directive is a marker on an HTML element (such as an attribute, element name, comment, or CSS class) that AngularJS recognizes and uses to create a customized set of functionality and/or user interface.
- AngularJS comes with many pre-built directives, but you can also write your own custom directives using the app.directive

```
<form>
   <p>
      Enter your name:
      <input type="text" ng-model="name" required>
      <button type="button" ng-click="submit()" ng-disabled="!name">Submit
      </button>
   </p>
   <p ng-show="name">
      Hello, {{name}}
   </p>
</form>
```

Directives

- Some angular directives
  - The ng-app - Bootstrapping your app and defining its scope.
  - The ng-controller - defines which controller will be in charge of your view.
  - The ng-repeat - Allows for looping through collections
  - ng-show directive shows or hides an HTML element
  - ng-click directive allows to specify a custom behavior when an element is clicked.
  - ng-disabled sets the disabled attribute on the element if the expression inside ngDisabled evaluates to true

Scope

- A scope is the context in which an expression is evaluated.
- This example has three scopes, one of which inherits a variable from its parent scope.

```
function foo() {
   var name = "John";

   function hello() {
      var name = "Jack";
      return "Hello, " + name;
   }

   function goodbye() {
      return "Good bye, " + name;
   }

   hello();   //returns "Hello, Jack"
   goodbye(); //returns "Good bye, John";
}
```

Service

- A service is the supplying or supplier of utilities or commodities, as water, electricity, or gas, required or demanded by the public. (http://dictionary.reference.com/browse/service)
- In programming is defined as …
  - The supplying of supplier of utilities or commodities, as **functions**, **values**, or **objects**, required or demanded by **an expression**.
- In AngularJS you can define services that provide functionality that can be used repeatedly throughout your code.

Dependency Injection

- Dependency injection is a design pattern that implements something called inversion of control for resolving dependencies
  - Client gets called with the dependencies by the surrounding angular js ecosystem is
- The whole point is that the client is not responsible for instantiating the dependency of the code that it depends on.

Reusable functionality

MODULES

Modules

- A module is a container for code for the different parts of your applications.
- Created using the AngularJS function `angular.module`
- A module is used to define **services** that are reusable by both the HTML document and other modules:
  - Controller
  - Directive
  - Constant, Value
  - Factory, Provider, Service
  - Filter
- **Best Practice:** Divide your code into modules with distinct functionality. Don't put everything in one module.

Module Definition

- Define a module:

```
var module = angular.module('myModule', []);
```

- The "myModule" parameter refers to an HTML element in which the application will run.
- Define a module with dependencies on other modules:

```
var module = angular.module('myModule', ['otherModule']);
```

- Get an existing module:

```
var module = angular.module('myModule');
```

Application Module

- AngularJS provides a way for you to bind your main module to the HTML document using the ng-app directive.

HTML fragment

```
<div ng-app='myApp'>
   ...
</div>
```

Javascript fragment

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

Module Phases

When a module runs it has two phases that you can link into.

Config

- The config phase happens early while the application is still being built.
- Only the provider services and constant services are ready for dependency injection at this stage.

RUN

- The run phase happens once the module has loaded all of its services and dependencies.

Module Phases

```
var module = angular.module('myModule', []);

module.config([function() {
   alert('I run first');
}]);

module.run([function() {
   alert('I run second');
}]);
```

Module Components and Dependency Injection

- AngularJS lets you inject **services** (either from its own module or from other modules) with the following pattern:

```
var module = angular.module('myModule', []);

module.service('serviceA', function() { ... });

module.service('serviceB', function(serviceA) { ... });
```

- What about JavaScript minifiers?

```
module.service('serviceB', ['serviceA',
function(serviceA) {
   ...
}]);
```

Facilitating Communication between the Model and the view

CONTROLLERS

Controller Definition and Assignment

Controllers link the model and the view using the AngularJS service: $scope
Nesting controllers is both possible and frequently done.

HTML Fragment

```
<div ng-app='myApp' ng-
controller="myController">
   <p>Hello, {{name}}!</p>
   <p>{{greet()}}</p>
</div>
```

JavaScript fragment

```
var module = angular.module('myApp', []);

module.controller('myController', [
   '$scope', function($scope) {

      $scope.name = 'John Smith';

      $scope.greet = function() {
         return 'Hello, ' + $scope.name + '!';
      };

   }]);
```

Changing the way you see things

FILTERS

Filters

- A filter formats the value of an expression for display to the user. (https://docs.angularjs.org/guide/filter)
- Filters can be used in HTML using the bar notation or they can be used in JavaScript by injecting the $filter service.

Filters

HTML Example

```
<div ng-app='myApp' ng-controller="myController">
   <p>{{name | uppercase}}</p>
   <p>{{uppercaseName()}}</p>
</div>
```

JavaScript Example

```
var module = angular.module('myApp', []);

module.controller('myController', [
   '$scope', '$filter', function($scope, $filter) {

      $scope.name = 'John Smith';

      $scope.uppercaseName = function() {
         return $filter('uppercase')($scope.name);
      };

   }]);
```

Filters with Parameters

HTML Example

```
{{ expression | filterName : param1 : param2 }}
```

JavaScript example

```
$filter('filterName')(expression, param1, param2);
```

Core Filters

- AngularJS has several filters built in:
  - currency
  - date
  - filter
  - json
  - limitTo
  - lowercase
  - number
  - orderby
  - uppercase

```
{{ '2015-03-19T19:00:00.000Z' | date : 'MMMM yyyy' }}
```

```
$filter('date')(new Date(), 'MMMM yyyy');
```

Core Filters

- AngularJS has several filters built in:
  - currency
  - date
  - filter
  - json
  - limitTo
  - lowercase
  - number
  - orderby
  - uppercase

```
{{ ['pear', 'apple', 'orange'] | filter : 'r' }}
```

```
$filter('filter')(['pear', 'apple', 'orange'], 'r');
```

Core Filters

- AngularJS has several filters built in:
  - currency
  - date
  - filter
  - json
  - limitTo
  - lowercase
  - number
  - orderby
  - uppercase

```
{{ { first: 'John', last: 'Smith } | json }}
```

```
$filter('json')({ first: 'John', last: 'Smith'});
```

Filter Definition

- You can define a new filter within a module.

```
angular.module('reverseFilter', [])
   .filter('reverse', [function() {
      return function(input, uppercase) {
         var i, out = "";
         input = input || '';
         for (i = 0; i < input.length; i++) {
            out = input.charAt(i) + out;
         }
       // conditional based on optional argument
         if (uppercase) {
            out = out.toUpperCase();
         }
         return out;
      };
   }]);
```

```
{{ 'Hello, World!' |
reverse : true }}
```

```
$filter('reverse')('Hello
, World!', true);
```

Five Recipe Flavors

SERVICES

Value

- The value recipe stores a value within an injectable service.
- A value can store any service type: a string, a number, a function, and object, etc.
- This value of this service can now be injected into any controller, filter, or service.

```
//define a module
var myModule = angular.module('myModule', []);

//define a value
myModule.value('clientId', 'a12345654321x');

//define a controller that injects the value
myModule.controller('myController', ['$scope', 'clientId', function
($scope, clientId) {
   $scope.clientId = clientId;
}]);
```

Factory

- The factory recipe is similar to a value recipe except that it adds these abilities:
  - Ability to use dependency injection
  - Lazy initialization
- A factory (like a value) can also return any data type.

```
//define a factory
myModule.factory('toUpperCase', ['$filter', function($filter) {

   return function(value) {
      return $filter('uppercase')(value);
   }

}]);

//inject the toUpperCase factory
myModule.controller('myController', ['toUpperCase', function(toUpperCase) {
   this.uppercase = toUpperCase('john');
}]);
```

Service

- Just to make things a little confusing, we have a service recipe called service.
  - Yes, we have called one of our service recipes 'Service'. We regret this and know that we'll be somehow punished for our misdeed. It's like we named one of our offspring 'Child'. Boy, that would mess with the teachers. (https://docs.angularjs.org/guide/providers)

Service

- The service recipe will generate a singleton of an instantiated object.

```
//define a service
myModule.service('person', [function() {
   this.first = 'John';

   this.last = 'Jones';

   this.name = function() {
      return this.first + ' ' + this.last;
   };
}]);

//inject the person service
myModule.controller('myController', ['$scope', 'person', function($scope, person) {
   $scope.name = person.name();
}]);
```

Provider

- "…the Provider recipe is the core recipe type and all the other recipe types are just syntactic sugar on top of it. It is the most verbose recipe with the most abilities, but for most services it's overkill" (https://docs.angularjs.org/guide/providers).
- The provider recipe can be injected during a module's configuration phase.

Provider

```
//define a provider
myModule.provider('prefix', [function() {
   var prefix = '';

   //setPrefix can be called during the module's config phase
   this.setPrefix = function(value) {
      prefix = value;
   };

   //this special property is required and returns a factory
   this.$get = [function() {
      return function(value) {
         return prefix + value;
      }
   }];
}]);

//inject the provider in the config phase
myModule.config(['prefixProvider', function(prefixProvider) {
   prefixProvider.setPrefix('John ');
}]);

//inject the provider's factory
myModule.controller('myController', ['prefix', function(prefix) {
   this.value = prefix('Smith');  //value is set to "John Smith"
}]);
```

Constant

- The constant recipe is similar to the value recipe except that its service value is also available during the module's configuration phase.

```
myModule.constant('author', 'John Smith');

myModule.config(['author', function(author) {
   console.log('This app was made by: ' + author); // "John Smith"
}]);

myModule.controller('myController', ["author", function (author) {
   this.author = author; // "John Smith"
}]);
```

Extending HTML

DIRECTIVES

Directives

- Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element or even transform the DOM element and its children. (https://docs.angularjs.org/guide/directive)

```
<form>
   <p>
      Enter your name:
      <input type="text" ng-model="name" required>
      <button type="button" ng-click="submit()" ng-
disabled="!name">Submit</button>
   </p>
   <p ng-show="name">
      Hello, {{name}}
   </p>
</form>
```

Directives

- At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element or even transform the DOM element and its children. (https://docs.angularjs.org/guide/directive)
- Writing custom directives is not for the faint of heart.

Directive Documentation

- For full documentation on how to write directives, see these pages:
- https://docs.angularjs.org/guide/directive
- https://docs.angularjs.org/api/ng/service/$compile

Directive Naming

- When defining a directive in JavaScript, the name is in camel case format:

```
module.directive('myDirective', [function() { ... }]);
```

- When we activate that directive we use a lower case form:

```
<my-directive></my-directive>

<div my-directive></div>
```

- AngularJS will normalize the HTML to match directive names by:
  1. Stripping the x- and data- from the front of the element/attributes
  2. Converting the colon, dash, and underscore delimited name into camel case
     - <div data-my-directive> is recognized as the directive myDirective

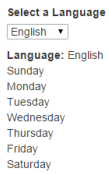
```
myModule.directive('directiveName', [function() {
   return {
      priority: 0,
      template: '<div></div>', // or // function(tElement, tAttrs) { ... },
      // or
      // templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
      transclude: false,
      restrict: 'A', // "AEMC"
      templateNamespace: 'html', // 'html', 'svg', 'math'
      scope: false, // true, {}
      controller: function($scope, $element, $attrs, $transclude, otherInjectables) { ...
},
      controllerAs: 'stringAlias',
      require: 'siblingDirectiveName', // or // ['^parentDirectiveName',
'?optionalDirectiveName', '?^optionalParent'],
      compile: function compile(tElement, tAttrs, transclude) {
         return {
            pre: function preLink(scope, iElement, iAttrs, controller) { ... },
            post: function postLink(scope, iElement, iAttrs, controller) { ... }
         }
         // or
         // return function postLink( ... ) { ... }
      },
      // or
      // link: {
      //  pre: function preLink(scope, iElement, iAttrs, controllers) { ... },
      //  post: function postLink(scope, iElement, iAttrs, controllers) { ... }
      // }
      // or
      // link: function postLink( ... ) { ... }
   };
}]);
```

Words in action

DEMO APPLICATION

Demo Application

- https://wp-dev.byu.edu/example



Example Application: week.js

```
//define the "week" module
var module = angular.module('week', []);

//define the daysOfWeek provider
module.provider('daysOfWeek', [function() {
   var languages, defaultLanguage;

   //define default languages
   languages = {
      English: [ 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday' ],
      French: [ 'dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi' ],
      German: [ 'Sonntag', 'Montag', 'Dienstag', 'Mittwoch', 'Donnerstag', 'Freitag', 'Samstag' ],
      Spanish: [ 'domingo', 'lunes', 'martes', 'miercoles', 'jueves', 'viernes', 'sabado' ]
   };
   defaultLanguage = 'English';

   //define the days of a week for a language
   this.define = function(language, sun, mon, tue, wed, thu, fri, sat) {
      languages[language] = [ mon, tue, wed, thu, fri, sat ];
   };

   //get or set the default language
   this.defaultLanguage = function(language) {
      defaultLanguage = language;
      return defaultLanguage;
   };

   //the factory
   this.$get = [function() {
      var factory;
      factory = {
```

Ex

```
   //define the days of a week for a language
   this.define = function(language, sun, mon, tue, wed, thu, fri, sat) {
      languages[language] = [ mon, tue, wed, thu, fri, sat ];
   };

   //get or set the default language
   this.defaultLanguage = function(language) {
      defaultLanguage = language;
      return defaultLanguage;
   };

   //the factory
   this.$get = [function() {
      var factory;
      factory = {

         //get an array of the days of the week for the language specified
         days: function(language) {
            if (arguments.length === 0) language = factory.defaultLanguage;
            if (!(language in languages)) throw new Error('Unknown language specified');
            return languages[language];
         },

         //the default language
         defaultLanguage: defaultLanguage,

         //get available languages
         languages: function() {
            return Object.keys(languages);
         }

      };
      return factory;
   }];
}]);

//create a days of week directive to print out the days of the week in a specified language
module.directive('daysOfWeek', ['daysOfWeek', function(daysOfWeek) {
   return {
      restrict: 'A',
      scope: {
         language: '=daysOfWeek'
      },
      template: '<span ng-repeat="day in days()">{{day}}<br></span>',
      link: function(scope, el, attrs) {
```

Example Application: app.js

```
//define our application and require the "week" module
var app = angular.module('app', ['week']);

//define a new language for the days of the week service
app.config(['daysOfWeekProvider', function(daysOfWeekProvider) {

   //define the days of the week in Danish
   daysOfWeekProvider.define('Danish', 'sondag', 'mandag', 'tirsdag', 'onsdag',
'torsdag', 'fredag', 'lordag');
}]);

app.controller('appController', ['$scope', 'daysOfWeek', function($scope,
daysOfWeek) {

   //set the selected language
   $scope.language = daysOfWeek.defaultLanguage;

   //get the list of languages
   $scope.languages = daysOfWeek.languages();

}]);
```

Demo Application: index.html

```
<!doctype html>
<html lang="en">
   <head>
      <title>App Title</title>
   </head>
   <body>
      <div ng-app="app" ng-controller="appController">
         <p>
            <label for="Languages">Select a Language</label><br>
            <select id="Languages" ng-options="label for label in languages" ng-
model="language"></select>
         </p>
         <p>
            <strong>Language: </strong> {{language}}<br>
            <span days-of-week="language"></span>
         </p>
      </div>

      <script src="js/angular.js"></script>
      <script src="js/app.js"></script>
      <script src="js/week.js"></script>
   </body>
</html>
```

The end is nigh

CONCLUSIONS

Essentials for Building Your First App

HTML Document

```
<!doctype html>
<html lang="en" ng-app="app">
   <head>
      <title>App Title</title>
      <link rel="stylesheet" href="css/styles.css">
   </head>
   <body>
      <script src="js/angular.js"></script>
      <script src="js/app.js"></script>
      <script src="js/moduleA.js"></script>
   </body>
</html>
```

APP.js

```
var module = angular.module('app', ['moduleA']);
```

Great Resources

- Official Tutorial – https://docs.angularjs.org/tutorial
- Official API – https://docs.angularjs.org/api
- Developer Guide – https://docs.angularjs.org/guide
- Video Tutorials – https://egghead.io/
- Video Introduction – https://www.youtube.com/watch?v=i9MHigUZKEM
- YouTube Channel – https://www.youtube.com/channel/UCm9iiIfgmVODUJxINecHQkA
- Articles, explanations, tutorials – http://www.ng-newsletter.com/