AngularJS — 11/2-11/5/15

instructor: John Paxton, pax@speedingplanet.com

Day 1 — 11/2/15

Setup (macs)

- > cd AngularClass/bin
- > source set-path
- > start-server
- > mongo-start
- > mongo-load-class (only the first time)
- > start-rest

On using “data-“ prefix - Totally personal preference. If you need to pass HTML 5 validation do it.

- for {{ }}… typically you’ll only use them for displaying text on screen but think of them as whenever you need to render the text aka value=“{{ }}”

ng-model

- for 2-way data binding
    - if one (model or view) updates, so does the other
- ng-model-options — provides add-on options such as delay update
    - ng-model-options=“{debounce: 1500}”
- If just using ng-model, the JS object won’t exist until you enter data

Following evaluate to false in JS (same for AngularJS 1.4+)

- NaN
- false
- “” or ‘’
- undefined
- null
- 0
- EVERYTHING ELSE evaluates to true

Chrome hack for console — When you inspect an element, chrome stores a reference to it in your console under $0

- angular.element($0).scope() will give you access to the angular scope involving that element

AngularJS + JQuery

- They work well but you need to include JQuery first!
- AngularJS can detect JQuery and then include or not include JQLite if it’s not there

Controllers

- Have all JS code inside IFE (Immediately invoked expression)
    - (function (angular) { }) (angular);
- Angular Modules are the repositories for all angular code
    - somewhat like java packages and/
- Controllers are the glue that holds models and view together
- only one ng-app will only be tied to one module
    - but that module could have dependancies
    - ng-app="firstApp" is basically "load the module"

Help compare frameworks —> www.todoMVC.com

- "==" allows for type coersion. "===" compares types and values
- Anytime

Angular expression advantages

- more error tolerant
    - Anytime an error occurs inside of {{ }}, an empty string is rendered and the error is pushed to the console. out of sight for the user

Filters

- there is a filter called filter
- take input and apply the filter to the input
    - orderBy, date, number, currency…
- Available at View AND Controller level
- as filteredTeams acts as an alias for the results
- During an ng-repat: The order of operation are
    - Filters
    - alias (as)
    - repeat over results
- Couple of options for custom filtering
    - passing a custom function to filter filter
        - not recommended b/c you can't pass arguments and will be tightly coupled to view
    - create a custom filter
        - better b/c you can pass multiple, colon-delimited arguments
        - The first argument will always be the object/string that is getting filtered (i.e. array that you're looking for)
    - ex in AngularClass/IntroAngular/Exercises/iterate-data
- Filtering objects
    - Default behavior - string will get tested on every property in object and the only way it will be tossed out if it fails on EVERY property
        - It will automatically pass on the first one it sees as true (quick to pass, slow to fail)

    - If you want to filter on a specific property in an object. Make your filtering string into a filtering object that has the property you want to filter on. (e.g., fPerson.name)
        - ex in /IntroAngular/Demos/multiple-filters

Angular Events

- same events as HTML but proceeded by ng-
- ng-click and $watch can accomplish the same functionality. $watch is faster b/c it doesn't use the DOM

ng-options

- shortcut for repeated over multiple options in <select></select> element as opposed to the more robust ng-repeat
- syntax = thing.label for thing in collection
- You can give ONE hardcoded object with the empty string value attribute. This will be the default value selected on page load

———

Day 2 — 11/3/15

Testing

- Angular was built from the ground up with testing in mind
- Test-framework agnostic but most tend towards TDD
    - Default usage is **Jasmine**
- Angular team built a generic test runner which doesn't require AngularJS
    - **Karma** — Built on top of Node JS (Originally called Testacular)
        - karma init filename — Simple way to initialize a config file
- Gulp would act as the task-manager in your process (a step back from karma)
    - It would call karma test, if pass deploy to github, etc…
- **Jasmine** —> unit test library
    - another one would be Mocha (very similar but typically preferred by Node.js ppl)

```
    describe(msg, fn() ({  // Multiple describes
          it(msg, fn() {    // What actually gets tested. Can have multiple in
describe functions
                expect(actual.toBe(expected)
          })
    })
```

- **Angular Mocks**
    - module — use this in beforeEach to load a particular module
    - inject — Runs Angular's injector service on a provided function, allowing you to control what is loaded when
- **Protractor — End-to-End testing**

    - Wrapped library around the widely-used UI testing software, **Selenium**
    - Install:
        - npm install protractor -g
        - protractor —version
    - Framework can be: Only Jasmine and Jasmine2 are officially supported
        - jasmine, cucumber, mocha, jasmine2

Ajax requests with $http

var p = $http({url: '/movies/1', method: 'get'})

p.success(function (data, status, headers, config)

- This was deprecated in Angular 1.4 b/c promises are coming to Angular in ES6

instead use:

```
p.then( function( retObj ) {
      $scope.movie = retObj.data;
} );
```

- retObj will then have the same four properties under it

Convenience

- ng-class
    - accepts several different types of expressions and maps of KVPs where the key is the class name and the value is the expression to be evaluated
        - The class will be applied if the value expression in the KVP evaluates to true
- Minification
    - Goal is to reduce the total number of characters in your code (including naming references in your method)
    - However, to get around this, String are NEVER minified
    - NEW SYNTAX: 2nd argument in controller now becomes an array and so the arguments in the function call can be matched up by position instead of name…it's okay to rename them now

```
someApp.controller('SomeCtrl', ['$scope', '$http',
      function($scope, $http) {
            // Do whatever with the controller here
      }
]);
```

    - ng-annotate — Tools that will help with this syntax rewriting
        - It will tell you where you don't pass manification
        - It can fix them in place
        - you can plug this into gulp to have it automatically done for you
-

MORE AJAX

- $http — has promises for dealing with Ajax

    - .then(onSuccess, onFailure) is how you register promise your code on a request object
        - arguments to onSuccess/onFailure
    - p.catch(onFailure) === p.then(null, failure)
    - p.finally(callback) — fires whether the Promise completed successfully or not. but no data is passed in.
- $http options
    - params — appended to the URL
    - data — into the body of the request
    - headers — map of headers and values, values can be strings or functions that return strings (evaluated at request time)
    - timeout — in milliseconds
    - withCredentials — send credentials with this request

Caching
- add **cache: true** to the options passed to $http
- If you make a request to the same url again and the response is still around, angular will use the cached response
- $cacheFactory
    - only will re-request if you invalidate the cache by .remove(url) or .removeAll()

$scope
- $scope's inherit from each other in a hierarchical form (think class inheritance in Java)
- they do NOT inherit laterally to siblings
- you can force it to go back up by using $parent as many times as you want (there's also $rootScope)
- This introduces some confusion and mystery
    - Angular development is starting to move away from this

——

Day 3 — 11/4/15

Recap So Far

View
- ng-app
- ng-controller
- ng-model
- ng-if
- ng-hide/ng-show
- ng-class/ng-style
- ng-repeat
    - filters, ordering, formatting

- ng-options
- ng-click, etc

<u>Tools</u>
- $http
- $log
- $scope

Other
- ng-annotate
- Karma + Jasmine + Angular
- Protractor

$http
- alternative to the way we've seen so far
    - var promise = $http.get('/data/movies', { params : _______ , timeout : _______ , withCredentials : _______ });
        - .post, .put, etc.

Routing
- 2 Different Options
    - Stock Router (thought of as being inferior)
        - Doesn't allow for granular control of view changes (think paneled application)
    - UI Router — Better option
        - Allows for as granular control as you'd like
        - C1 suggests you should **prefer** UI Router
- Setting up
    - HTML
        - <script>angular-ui-router.js</script>
        - ui-view — The target div to be changed
        - ui-sref — Switch between states/route
    - JS
        - Module depends on ui.router
            - var mod = angular.module('uiRouterMod', ['ui.router', 'uiRouterControllers']);
                - ui.router — strictly required
                - uiRouterController —
        - Configure module with states
            - mod.config(function ($stateProvider, $urlRouterProvider) { … });
                - $stateProvider and $urlRouterProvider are handled by the injector and are provided by UI Router
        - For any given module there are two phases that are only really relevant for routing
            - Config — ALL routing should be registered during the config phase
                - mod.config is the hook to do this

            - Run
- DAO — Data Access Object
    - Lazy-loading. When the templates are loaded there is no data until you its run
- passing variables to parameters
    - pull off in $stateParams
    - listed that a variable is expected in your config (app.js) by : or { }
    - e.g., /:productID or /{productID}
- # comes from browsers not removing your resources after #
    - it is an event that can be captured and the behavior changed
    - the router captures the event, tells it not to try and scroll, instead load in the resources
    - Modern browsers don't need it and it can be turned off

Providers

- $http is a provider
- Concept is basically analogous to an API. You can create them to handle common actions across multiple controllers (aka getting data — DAO)
- Basically Angular's Objects
    - Provider (parent class)
        - Value — Allows you to create a value that is available application wide
            - e.g., mod.value('foo', 'bar');
                - value is called foo and set to bar
            - Primitive objects are immutable
            - References are immutable but the objects inside the reference are not
            - typically used for API keys
        - Constant — VERY similar to the value provider
            - Only major difference is that a constant is available earlier in the module's lifecycle
            - Constants are available at the config phase of a module's lifecycle, whereas values (and factories and services) are only available at run phase
            - There isn't a negative to using only constants rather than values
        - Factory — More customizable than Values
            - They can have dependencies
            - Can initialize the provide
            - Initialization is deferred until it is needed (as opposed to immediate initialization for Values)
            - Can create objects of various types. However it only generates the object once b/c it's a singleton
        - Service — Basically a factory but always returns an object
            - Use factory or service…whichever you want
- ALL providers are singletons
- **Angular encourages holding onto references when dealing with factories b/c you can then watch the reference and will pick up when the data has changed and will re-render the page**
- **Underscore.js** — Helpful utility toolset to keep on hand

——

Day 4 — 11/5/15

The State Machine

- Finite number of states that your routing can be in (think Stoplight…three states)
- e.g., Customers
    - customers.detail
    - customers.add
    - customers.edit
    - customers.search
- $routeProvider vs $stateProvider (uiRouter)
    - $routeProvider
        - when /customers
        - To change the way route works you'll have to find and replace all references to "/when" and replace them with new url
            - <a href="/customers/ABCDE">…</a>
    - $stateProivider
        - state 'customers'
        - To change the way route works you'll only have to change the url under the state config. You don't have to change anything else b/c you've been referring to the state
            - <a ui-sref="customer.detail({customers:ABCDE})">…</a>
        - with ui-sref you can also tell it to generically go to the sibling or child view without specifying the parent
            - e.g., <a ui-sref="^.edit">…</a> — Go to sibling view
                - ^ = Parent

Multi-views with $stateProvider

- single url
- then a json object with templateUrl's and controllers

Directives!

- Default behavior is that it can be either attribute or element but you can restrict it
    - You can use restrict A, E, and/or C to restrict to attribute, element, or class definitions (class is deprecated)
- html naming is hyphenated, js name is camel case
- scope declaration in directives give you an isolate scope to work with.
    - these scopes do not participate in the scope hierarchy at all

      JS

      ```
      scope : {
          firstName : '@yourName'
      }
      ```

        HTML
```
<greet-person your-name=“Adam”></greet-person>
```
    - left hand side in the declaration is the name of the variable in the template and the right hand side is the attribute name that will pass the variable through
    - Typically you want the variables to be the same name
        - short hand declaration if you want them to be the same thing

          JS
```
scope : {
        firstName : ‘@‘
}
```
    - further shorthand… using ‘=‘ allows you to pass the variable value from the scope through instead of the literal value so you don’t have to use the {{ }} in the attribute
- **You can kind of think of directives like functions for HTML**
- **Routing vs. Directives vs. ng-includes**
    - routing is like changing scenes in a play
    - directives are like the actors in the play
    - ng-includes are like the props
- Link function
    - Allows you to manipulate the DOM from directives
- **$location.path()** — get everything in the url
- angular has a lot of normal HTML directives that it overrides
    - a, form, etc…
    - they automatically give you access to controllers, scope, etc and helpful variables
    - <form name=“personForm”> ==> FormController
        - $scope.personForm is a reference to the FormController off of which you can get properties
        - <input type=“text” name=“firstName” ng-model=“firstName”>
            - $scope.personForm.firstName — current state of the field
            - $scope.firstName — Current value of the field
- You can have directives talk to each other through controllers using the require statement
-