Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311

AngularJS Directives

Software Craftsmanship Guild

Lesson Goals

Introduce the basic concepts behind creating your own AngularJS directives.

Angular Directives

Angular makes it relatively easy to add our own directives for use in our applications. Directives can have their own scope, can build complex html templates, and can define events. An example of a powerful directive is ng-repeat.

Writing your own directives is pretty advanced stuff, you'll need a strong working knowledge of HTML and JavaScript to build interesting things.

Example: Movie Ratings Directive

Let's say that we want to create a directive for our movie application that allows uses to dynamically select a rating based on a maximum number of stars provided.

It should support choosing a rating by clicking a star, hovering over stars should animate nicely, and the label should be able to be changed via attributes.

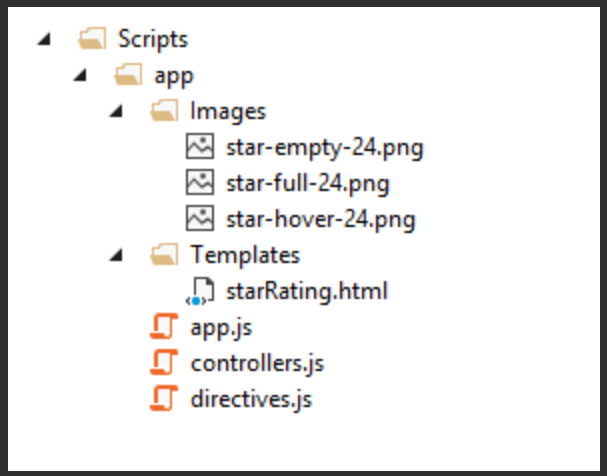
The end result should be something like this:

Setup

Let's create an empty ASP.NET MVC project (MVC and WebApi), update NuGet, and add AngularJS Core +Routing, and Bootstrap.

Add a Home Controller with an Index and generate a view with a shared layout.

Lastly, create an app section in our scripts folder and add some necessary files:



Declaring a new directive

In order to declare a new directive, we have to attach it to an angular module.

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

A directive must return an object to the app. The properties of the object are pre-defined by the angular documentation. Let's start with 'restrict'

```
app.directive('ngStarRating', function () {
    // directives return an object
    return {
        restrict: 'EA', // declare as element or attribute
```

Breaking it down

```
app.directive('ngStarRating', function () {
    // directives return an object
    return {
        restrict: 'EA', // declare as element or attribute
```

The name of the directive is ngStarRating. Angular has some magic to add dashes between 'words' so in your HTML we will say ng-star-rating.

For the restrict we passed a text value of EA which means Element or Attribute.

* generally we shouldn't use the ng prefix though because that is what angular libraries use.

Restrict Values

Value	Description	Example
E	Directive as element	<ng-star-rating></ng-star-rating>
A	Attribute on existing element	<div ng-star-rating>
C	CSS class on existing element	<div class="ng-star-rating>
M	Directive applied as comment	<!-- directive: ng-star-rating -->

* CSS and Comment directives are not best practices in the Angular community. They were created for some backwards browser compatibility purposes.

Declaring a Scope and Controller

We could potentially have multiple rating pickers on the same page, so in this case we will want to give our directive its own scope and it's own controller.

Let's start with the Scope.

Isolating Scope

Oftentimes we want to ensure that our directive has its own scope, especially when we are manipulating many directives on the same page. However, we also often want to be able to bind values from other controllers into our directive.

AngularJS provides flexibility to selectively pass parent scope properties to our directive through bindings.

For our star rating directive, we will want to bind the selected rating to the page's controller. We will also provide some flexibility by letting the HTML determine how many maximum stars to show.

One-Way Binding With @

When we declare options in our scope, we can use the @ symbol for one-way binding. This allows for custom HTML attributes with values to pass data into the directive.

Let's allow the author of the HTML to specify the maxRating in stars.

```
app.directive('ngStarRating', function() {
    return {
        restrict: 'EA', // element or attribute
        replace: true,
        templateUrl: "/Scripts/app/Templates/starRating.html",
        scope: {
            maxRating: '@', // attribute
        }
    };
});
```

Two-Way Binding With =

Some values we want to share with the directive and the controller for our page. This allows us to point a value from our directive to a page controller and keep them in sync.

We can two way bind as many properties as we want so long as they have unique names. Let's add a rating property that we can share with our page controller. By specifying =rating the page controller can bind any of its $scope properties to the rating via attribute (which we will see later).

```
scope: {
    rating: '=rating',    // two-way bind
    maxRating: '@',       // attribute
}
```

Using & to Call Functions in Parent

Sometimes we want to allow the parent (page) controller to plug into our directive's functions in a similar manner to event bubbling. To do this, we simply declare a function name with an & symbol.

Let's allow our parent controller to get notification of a star being clicked:

```
scope: {
    rating: '=rating',    // two-way bind
    maxRating: '@',       // attribute
    click: "&",           // function
}
```

Compile Function

Now, even though the page author can set a maxRating attribute on our directive, they may forget to fill it in.

The compile function runs once before any cloning occurs (such as ng-repeat) so it is a good place to do some set up that does not involve scope (since it runs before ng-repeat, compile executes before a scope is created)

Compile does have access to the element and the attributes of the element. Let's write some code to check to see if a maxRating was provided and if it wasn't let's default it to a 5 star rating.

There is also a link function which does have a scope and can watch the DOM for changes and respond. In ng-repeat you get a link for each element.

```
app.directive('ngStarRating', function() {
    return {
        restrict: 'EA', // element or attribute
        replace: true,
        templateUrl: "/Scripts/app/Templates/starRating.html",
        scope: {
            rating: '=rating',    // two-way bind
            maxRating: '@',       // attribute
            click: "&",           // function
        },
        compile: function (element, attrs) {
            // if they don't provide a maxRating attribute, default it to 5
            if (!attrs.maxRating || (Number(attrs.maxRating) <= 0)) {
                attrs.maxRating = '5';
            };
        }
    };
});
```

Declaring a Template

We can add a template option to our object and write HTML in the directive directly, or we can use the templateUrl option to provide a path to the an HTML file (similar to route templates).

For anything but very simple HTML, you should use a templateUrl.

```
app.directive('ngStarRating', function() {
    return {
        restrict: 'EA', // element or attribute
        replace: true,
        templateUrl: "/Scripts/app/Templates/starRating.html",
        scope: {
```

Replace Option

The replace option specifies whether it will replace the HTML element upon which the directive is applied.

Notice these two directives that print out Hello World in H3 tags. Instead of the html being in a file (templateUrl), in this case we wrote our html in-line using the template option.

When we use them in our HTML notice how in the output the replace option removes the div entirely and replaces it with the H3 tag while the other embeds the template inside the containing div.

```
app.directive('helloMessageReplace', function() {
    return {
        restrict: 'EA',
        template: '<h3>Hello World!</h3>',
        replace: true
    };
});

app.directive('helloMessage', function () {
    return {
        restrict: 'EA',
        template: '<h3>Hello World!</h3>',
        replace: false
    };
});
```

```
<div hello-message-replace></div>
<div hello-message></div>
```

```
<h3 hello-message-replace>Hello World!</h3>
<div hello-message>
  <h3>Hello World!</h3>
</div>
```

starRating.html

Like any other AngularJS template file we can just write HTML as if we are in the page.

```
<div style="display:inline-block; margin: 0; padding: 0; cursor:pointer;"
     ng-repeat='idx in maxRatings track by $index'>
    <img ng-src='{{getStarPath($index)}}'
         ng-click='starClick($index + 1)'/>
</div>
```

This is a lot of stuff to digest, let's break it down.

ng-repeat by $index and directive controllers

Here's a new trick. If we want to keep track of which element we are on in an ng-repeat statement we can add 'track by $index'.

I have referenced a new property here called maxRatings. What we want is to render one star for each rating up to the max (so if ratings are out of 5 stars, it should display 5 stars).

Each star rating directive should have its own copy of maxRatings, so we need to give our directive its own controller!

```
<div style="display:inline-block; margin: 0; padding
     ng-repeat='idx in maxRatings track by $index'>
    <img ng-src='{{getStarPath($index)}}'
         ng-click='starClick($index + 1)'/>
</div>
```

```
},
controller: function($scope, $element, $attrs) {
    $scope.maxRatings = [];
    $scope.rating = 0;

    // add an element for each rating
    // (used in ng-repeat in the template)
    for (var i = 1; i <= $scope.maxRating; i++) {
        $scope.maxRatings.push({});
    };
}
```

Conditional Image Paths

ng-src allows you to bind image paths to controllers. You can bind them to controller properties or to the results of functions.

In our case, if the selected rating is less than the index of the star then we should show an empty star. Otherwise we want to show a full star. We will return the path to our star images.

Let's also add a $scope._rating variable as well. Later on we will add a hover effect so we need to be able to tell the difference between the selected rating and what rating they are hovering over.

```
<div style="display:inline-block; margin: 0; padding: 0; cursor:pointer;"
     ng-repeat='idx in maxRatings track by $index'>
    <img ng-src='{{getStarPath($index)}}'
         ng-click='starClick($index + 1)'/>
</div>
```

```
controller: function($scope, $element, $attrs) {
    $scope.maxRatings = [];
    $scope.rating = $scope._rating = 0;

    // add an element for each rating
    // (used in ng-repeat in the template)
    for (var i = 1; i <= $scope.maxRating; i++) {
        $scope.maxRatings.push({});
    };

    // if the rating is less than the index (position) of the star
    // we should display an empty star... otherwise the filled star
    $scope.getStarPath = function(index) {
        if ($scope._rating <= index)
            return "/Scripts/app/Images/star-empty-24.png";

        return "/Scripts/app/Images/star-full-24.png";
    };
}
```

Adding a click function

When the user clicks on a star, we want to set the rating to the value of the star we clicked (indexes start from zero, so we need to add one).

This function can also be referenced by the parent controller as we will see later.

```
<div style="display:inline-block; margin: 0; padding: 0; cursor:pointer;"
     ng-repeat='idx in maxRatings track by $index'>
    <img ng-src='{{getStarPath($index)}}'
         ng-click='starClick($index + 1)'/>
</div>
```

```
controller: function($scope, $element, $attrs) {
    $scope.maxRatings = [];
    $scope.rating = $scope._rating = 0;

    // add an element for each rating
    // (used in ng-repeat in the template)
    for (var i = 1; i <= $scope.maxRating; i++) {
        $scope.maxRatings.push({});
    };

    // if the rating is less than the index (position) of the star
    // we should display an empty star... otherwise the filled star
    $scope.getStarPath = function(index) {
        if ($scope._rating <= index)
            return "/Scripts/app/Images/star-empty-24.png";

        return "/Scripts/app/Images/star-full-24.png";
    };

    $scope.starClick = function (starValue) {
        $scope.rating = $scope._rating = starValue;
        $scope.click({ starValue: starValue }); // send to parent
    };
}
```

Let's Test It!

```
<h2>Index</h2>

<div ng-app="movieApp" ng-controller="MovieController">
    <div class="row">
        <div class="col-xs-6">
            <form role="form">
                <div class="form-group">
                    <label>Rating</label>
                    <div ng-star-rating rating="movieRating" click="ratingClick(starValue)"></div>
                </div>
            </form>
        </div>
    </div>
</div>

@section scripts
{
    <script src="~/Scripts/app/app.js"></script>
    <script src="~/Scripts/app/directives.js"></script>
    <script src="~/Scripts/app/controllers.js"></script>
}
```

Add a Movie Controller and bring it all together!

See how the click event from the directive can be registered to a controller function.

Index.cshtml

```
<div ng-star-rating rating="movieRating" click="ratingClick(starValue)"></div>
```

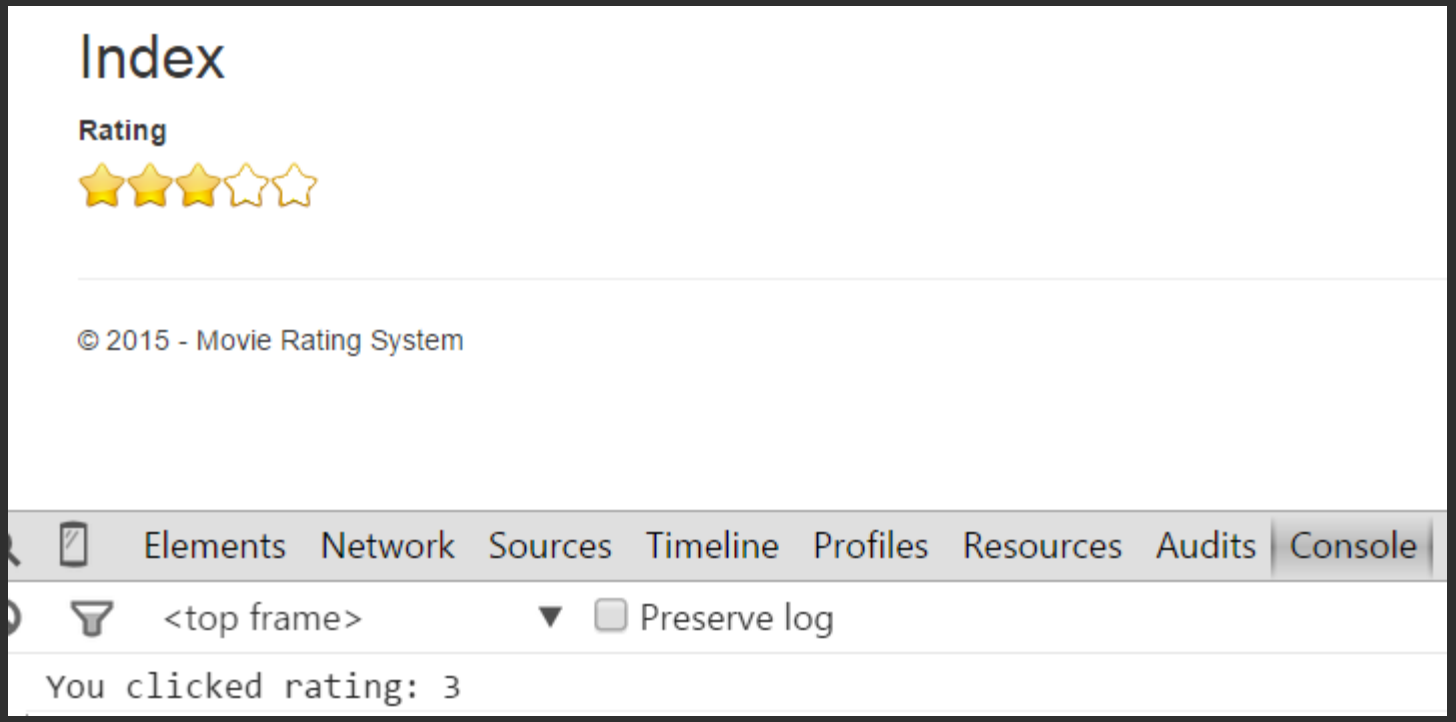
directives.js

```
app.directive('ngStarRating', function() {
    return {
        restrict: 'EA', // element or attribute
        templateUrl: "/Scripts/app/Templates/starRating.html",
        scope: {
            rating: '=rating',    // two-way bind
            maxRating: '@',       // attribute
            click: "&",           // function
        },

controller: function($scope, $element, $attrs) {
    $scope.maxRatings = [];
    $scope.rating = $scope._rating = 0;

            ...

    $scope.starClick = function (starValue) {
        $scope.rating = $scope._rating = starValue;
        $scope.click({ starValue: starValue }); // send to parent
    };
```

controllers.js

```
app.controller('MovieController', function ($scope) {
    $scope.ratingClick = function(starValue) {
        console.log('You clicked rating: ' + starValue);
    };
});
```

Success!


Index
Rating
© 2015 - Movie Rating System
Elements
Network
Sources
Timeline
Profiles
Resources
Audits
Console
<top frame>
Preserve log
You clicked rating: 3

Study Time!

Review the final version of the code in the MoviesSPA project which has been further enhanced to support hovering and a dynamic label.

Note how the progressive enhancements simply involved a few more data points and functions.

Conclusion

AngularJS directives can range from very simple to very complex. A thoughtful developer can even allow parent controllers to plug into the functions and variables of the directive.

A well made directive that can be consumed by the public takes a lot of planning and foresight to be successful.