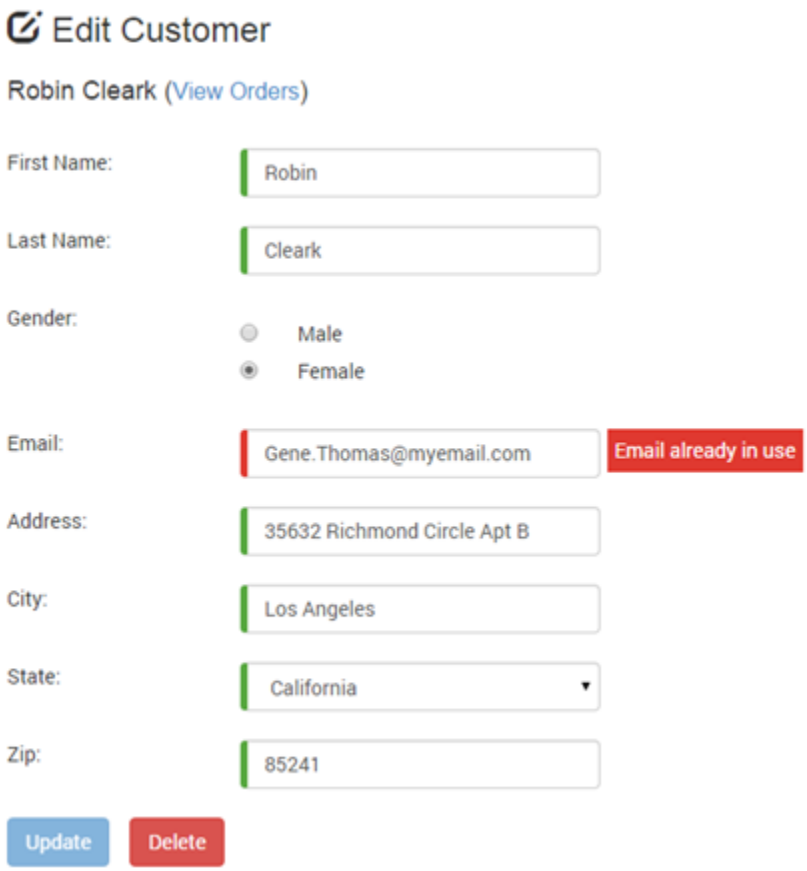
Creating Custom AngularJS Directives Part 7 – Creating a Unique Value Directive using $asyncValidators

by Dan Wahlin

In the previous article I demonstrated how to build a unique value directive to ensure that an email address isn't already in use before allowing a user to save a form. With changes in AngularJS 1.3+, several new features are available to clean up the previous version of the directive that I released on Github and make it easier to work with. In this article I'll walk-through some of the new features in a directive called **wcUnique,** and show how a few of the new features can be applied. The code shown is part of the Customer Manager Standard sample application that's available on Github. Note that more details on how to build this particular directive (if you'd like them - since it's more advanced) can be found in my AngularJS Custom Directives video course.

An example of the directive in action is shown next. In this example the email address shown is already in use by another user which causes the error message to be displayed.



The directive is applied to the **email** input control using the following code:

```
<input type="email" name="email"
        class="form-control"
        data-ng-model="customer.email"
        data-ng-model-options="{ updateOn: 'blur' }"
        data-wc-unique
        data-wc-unique-key="{{customer.id}}"
        data-wc-unique-property="email"
        data-ng-minlength="3"
        required />
```

You can see that a custom directive named **wc-unique** is applied as well as 2 properties named **wc-unique-key**and **wc-unique-property** (I prefer to add the **data-** prefix but it's not required). Before jumping into the directive let's take a look at an AngularJS factory that can be used to check if a value is unique or not.

Creating the Factory

Unique value checks typically rely on a call to a back-end data store of some type. AngularJS provides services such as **$http** and **$resource** that are perfect for that scenario. For this demonstration I'll use a custom factory named **dataService** that relies on **$http** to make Ajax calls back to a server. It has a function in it named**checkUniqueValue()** that handles the unique checks. Note that I don't typically distinguish between the term "factory" and "service" as far as the name goes since they ultimately do the same thing and "service" just sounds better to me (personal preference).

```
(function () {

    var injectParams = ['$http', '$q'];

    var customersFactory = function ($http, $q) {
        var serviceBase = '/api/dataservice/',
            factory = {};

        factory.checkUniqueValue = function (id, property, value) {
            if (!id) id = 0;
```

```
            return $http.get(serviceBase + 'checkUnique/' + id +
              '?property=' + property +
              '&value=' + escape(value)).then(
                function (results) {
                    return results.data.status;
                });
        };

        //More code follows

        return factory;
    };

    customersFactory.$inject = injectParams;

    angular.module('customersApp').factory('customersService',
      customersFactory);

}());
```

Creating the Unique Value Directive

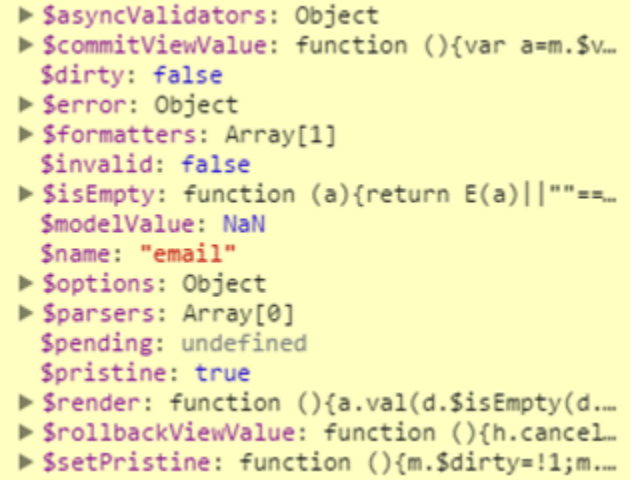
To handle checking unique values I created a custom directive named **wcUnique** (the wc stands for Wahlin Consulting – my company). It's a fairly simple directive that is restricted to being used as an attribute. The shell for the directive looks like the following:

```
function () {
```

```
    var injectParams = ['$q', 'dataService'];

    var wcUniqueDirective = function ($q, dataService) {
        return {
            restrict: 'A',
            require: 'ngModel',
            link: function (scope, element, attrs, ngModel) {

                //Validation code goes here

            }
        };
    };

    wcUniqueDirective.$inject = injectParams;

    angular.module('customersApp').directive('wcUnique',
      wcUniqueDirective);

}());
```

As the directive is loaded the **link()** function is called which gives access to the current scope, the element the directive is being applied to, attributes on the element, and the **ngModelController** object. If you've built custom directives before (hopefully you've been reading my series on directives!) you've probably seen **scope**,**element** and **attrs** before but the 4th parameter passed to the **link()** function may be new to you. If you need access to the **ngModel** directive that is applied to the element where the custom directive is attached to

you can “require” **ngModel** (as shown in the code above). ngModel will then be injected into the **link()** function as the 4th parameter and it can be used in a variety of ways including validation scenarios. One of the new properties available in **ngModelController** with AngularJS 1.3+ is **$asyncValidators** (read more about it here) which we’ll be using in this unique value directive.



Here’s the complete code for the **wcUnique** directive:

```
(function () {

    var injectParams = ['$q', '$parse', 'dataService'];

    var wcUniqueDirective = function ($q, $parse, dataService) {
        return {
            restrict: 'A',
            require: 'ngModel',
            link: function (scope, element, attrs, ngModel) {
```

```
                ngModel.$asyncValidators.unique =
                 function (modelValue, viewValue) {
                    var deferred = $q.defer(),
                        currentValue = modelValue || viewValue,
                        key = attrs.wcUniqueKey,
                        property = attrs.wcUniqueProperty;

                    if (key && property) {
                        dataService.checkUniqueValue(key, property,
                          currentValue)
                        .then(function (unique) {
                            if (unique) {
                                deferred.resolve(); //It's unique
                            }
                            else {
                                //Add unique to $errors
                                deferred.reject();}
                        });
                    }
                    else {
                        deferred.resolve();
                     }

                    return deferred.promise;
                };
            }
        };
    };

    wcUniqueDirective.$inject = injectParams;
```

```
    angular.module('customersApp').directive('wcUnique',
      wcUniqueDirective);


}());
```

You'll notice that the **ngModel** parameter that is injected into the **link()** function is used to access a property named **$asyncValidators**. This property allows async operations to be performed during the data validation process which is perfect when you need to go back to the server to check if a value is unique. In this case I created a new validator property named **unique** that is assigned to a function. The function creates a *deferred*object and returns the promise. From there the code grabs the current value of the input that we're trying to ensure is unique and also grabs the **key** and **property** attribute values shown earlier.

The **key** represents the unique key for the object (ultimately the unique identifier for the record). This is used so that we exclude the current object when checking for a unique value across objects on the server. The **property**represents the name of the object property that should be checked for uniqueness by the back-end system.

Once the variables are filled with data, the **key** and **property** values are passed along with the element's value (the value of the textbox for example) to a function named **checkUniqueValue()** that's provided by the**dataService** factory shown earlier. This triggers an Ajax call back to the server which returns a true or false value. If the server returns that the value is unique we'll resolve the promise that was returned. If the value isn't unique we'll reject the promise. A rejection causes the **unique** property to be added to the **$error** property of the **ngModel** so that we can use it in the view to show and hide and error message.

Showing Error Messages

The unique property added to the **$error** object can be used to show and hide error messages. In the previous section it was mentioned that the **$error** object is updated but how do you access the **$error** object? When using AngularJS forms, a **name** attribute is first added to the **<form>** element as shown next:

```
<form name="editForm">
```

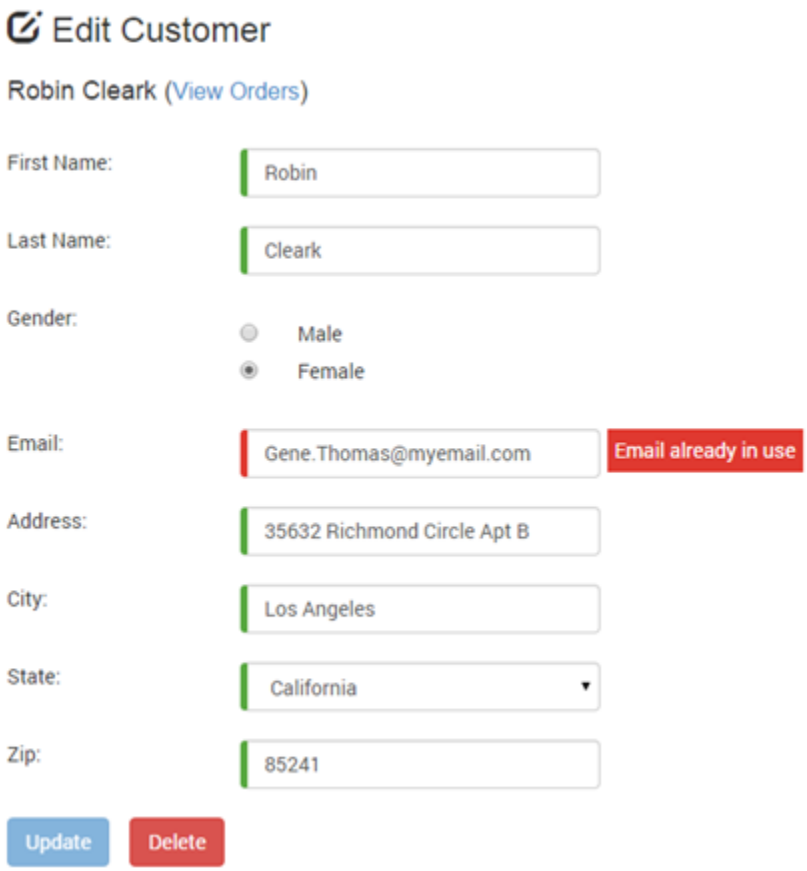
The **editForm** value causes AngularJS to create a child controller named **editForm** that is associated with the current scope. In other words, **editForm** is added as a property of the current scope (the scope originally created by your controller). The textbox shown earlier has a name attribute value of **email** which gets converted to a property that is added to the **editForm** controller. It's this **email** property that gets the **$error** object. Let's look at an example of how we can check the **unique** value to see if the email address is unique or not:

```
<div class="col-md-2">
    Email:
</div>
<div class="col-md-10">
    <!-- type="email" causing a problem with Breeze so using regex -->
    <input type="email" name="email"
           class="form-control"
           data-ng-model="customer.email"
           data-ng-model-options="{ updateOn: 'blur' }"
           data-wc-unique
           data-wc-unique-key="{{customer.id}}"
```

```
            data-wc-unique-property="email"
            data-ng-minlength="3"
            required />
    <!-- Show error if touched and unique is in error -->
    <span class="errorMessage"
    ng-show="editForm.email.$touched && editForm.email.$error.unique">
        Email already in use
    </span>
</div>
```

Notice that the **ngShow** directive (on the span at the bottom of the code) checks the **editForm** property of the current scope and then drills down into the **email** property. It checks if the value is touched using the **$touched**property (this property is in AngularJS 1.3+ and reports if the target control lost focus – it has nothing to do with a touch screen) and if the **$error.unique** value is there or not. If **editform.email.$error.unique** exists then we have a problem and the user entered an email that is already in use. It's a little bit confusing at first glance since we're checking if **unique** is added to the **$error** object which means the email is already in use (the unique property is in error). If it's not on the **$error** object then then everything is OK and the user entered a unique value.

The end result is the red error message shown next:



Note that starting with Angular 1.3 you can also use the ng-messages directive to show errors. More details about it can be found at https://docs.angularjs.org/api/ngMessages.

Conclusion

Directives provide a great way to encapsulate functionality that can be used in views. In this post you've seen a simple AngularJS unique directive that can be applied to textboxes to check if a specific value is unique or not and display a message to the user. To see the directive in an actual application check out the Customer Manager sample application at https://github.com/DanWahlin/CustomerManagerStandard.

Check out my AngularJS Custom Directives course for additional information about building your own directives.