Angular UI Sortable for Managing Sort Order

Posted At : July 13, 2014 11:28 PM | Posted By : Cutter
Related Categories: AngularJS

In my newest position I've been learning **AngularJS**. After diving in for a little over two months now, I'm seriously loving it, and wondering why I didn't seriously look at it much sooner.

One of the things it has taken a bit to get used to is that Angular does some things differently. I love **Bootstrap** but, like all other plugins, Angular's MVC approach has you do things "the Angular way", which can take a bit to break old habits. Luckily for us there are projects like **Angular UI** to help take the pain out of these sort of things.

One of the very few things that I use jQueryUI for is it's *Sortable*, *Draggable* and *Droppable* plugins. Angular UI has modules for these too. In this example, we're going to use the *Sortable* plugin for easy Drag n Drop display order management.

Using Drag n Drop for display order management is a fairly standard task. In the old days, we used up and down arrows that would typically be links, firing calls back to the server for database interaction and the refreshing the page for the new order. This was long, tedious, and a horrible user experience. Drag n Drop takes care of that. It allows the user to quickly define the order of things prior to ever sending data back to the server.

This demo isn't going to get into the server side mechanics of any of this, but rather the client-side stuff. First we'll start with the Bootstrap sample template (we'll use the Bootstrap CSS for basic layout stuff in our example). Don't forget to take out the Bootstrap JS file, and we're going to load the jQuery file up at the top ("why" will become clearer later).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap 101 Template</title>

    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->

     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
  </head>
  <body>
    <h1>Hello, world!</h1>

  </body>
</html>
```

A requirement of the Angular UI Sortable plugin is the actual jQueryUI *sortable* plugin, and it's dependencies. Luckily we can **download a really trim file**, giving us only what we need. We will insert the JS file in our document *head*, directly after our jQuery file.

```
<script src="/path/to/file/jquery-ui.min.js"></script>
```

After that, we drop in the CDN link for AngularJS:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-beta.14/angular.min.js"></script>
```

And then we need the **Angular UI sortable**. I haven't found a CDN for this one, so you'll have to install it locally:

```
<script src="/path/to/file/ui-sortable.js"></script>
```

And finally, let's create a file to put all of our application logic:

```
<script src="/path/to/file/sortable-app.js"></script>
```

In review, and inventory of our code show us that:

- We're loading the Bootstrap CSS
- We're loading some shivs, for handling HTML 5 stuff in IE 9 or earlier (good luck with that)
- We're loading jQuery, and our chopped down version of jQueryUI
- We're loading Angular, the Angular UI Sortable plugin, and a file for our application

Our app file is empty right now. Let's get it started by initializing our Angular application. We do this by implementing our app as an Angular *module*:

```
angular.module( "sortable", [ "ui.sortable" ] )
```

Basically this says our app is called "sortable", and it has a dependency on "ui.sortable". That part was easy enough, but now we have to associate the app with our display. We do this by modifying our opening *html* tag:

```
<html lang="en" ng-app="sortable">
```

The next thing I'll do is create a *controller* for the next part of our app.

```
.controller( "ListController", function ( $scope ) {

})
```

```
<body class="container" ng-controller="ListController">
```

We created a controller, and associated it with the *body* of our document. You can assign a controller to any element within your document, but this is a pretty simple example, so we're going simple here.

We need some data. Ordinarily you'd probably pull some data from the server, but for our demo we'll just create an array in our controller. The variable will be within the controller's *scope*, making it available to the block of our view within the ngController's elements.

```
$scope.bands = [
    {
     name: "The Beatles",
     rank: 1,
     mode: "view"
    },
    {
     name: "The Who",
     rank: 2,
     mode: "view"
    },
    {
     name: "The Jimi Hendrix Experience",
     rank: 3,
     mode: "view"
    },
    {
     name: "Led Zepplin",
     rank: 4,
     mode: "view"
    },
    {
     name: "The Doors",
```

```
     rank: 5,
     mode: "view"
    }
];
```

```
<body class="container" ng-controller="ListController">

<!--- The "bands" are available inside this block -->

</body>
```

Now that we can access our controller's variables, let's go ahead and output them to our screen:

```
<div class="container-fluid">
     <div ng-repeat="band in bands">
      <div ng-if="band.mode == 'view'" class="row well">
       <div class="col-xs-2">
        {{band.rank}}
       </div>
       <div class="col-xs-10">
        {{band.name}}
       </div>
      </div>
      <div ng-if="band.mode == 'edit'" class="row">
       <div class="col-xs-10">
        <input type="text" class="form-control" ng-model="band.name" placeholder="Band Name...">
       </div>
       <div class="col-xs-2 text-right">
        <button type="button" class="btn btn-primary" ng-click="addBand( band )">Add</button>
       </div>
      </div>
     </div>
    </div>
    <hr>
    <div class="text-right">
     <button ng-click="createBand()" class="btn btn-primary"><span class="glyphicon glyphicon-plus"></span></button>
    </div>
```

OK, I need to slow down a minute. Let's explain some code here. First, I loop over the *bands* for output. If it's in "view" mode, it shows you the *rank*, and the *name*. If it's in "edit" mode, then you see a text box and a button. Lastly, there's a button at the bottom for adding new bands.

There are two methods tied in here. Let's add them both to our controller:

```
$scope.createBand = function () {

 var band = {
  name: "",
```

```
  rank: $scope.bands.length,
  mode: "edit"
 };
 $scope.bands.push( band );

};

$scope.addBand = function ( band ) {

 band.mode = "view";

};
```

Nothing crazy here. We aren't doing any server-side stuff in this demo, so all we're trying to say here is "create" will put a stub object last in our *bands* array, while "add" will change a step's mode from "edit" to "view".

You can try that out now, if you want. Clicking on the "Plus" button at the bottom will add a new line for you. Type in a Band Name and then click that line's "Add" button. Each time you hit "Add", the mode changes to "view", and you see the display order followed by the Band name. You can add as many bands as you like, but so far we still can't re-order them, and anyone who knows me knows that as much as I love The Beatles, Jimi is number one in my book.

You'll remember that we already have the Angular UI Sortable plugin available to us. We included the code, and the dependency, now we just need to implement it. On the *div* that contains our *ngRepeat* loop directive, just add the directive call for the Angular UI Sortable:

```
<div ui-sortable ng-model="bands" class="container-fluid">
```

This adds the directive to the container, as well as defining the *model* involved. *ngModel* is critical here, or it won't work. If you reload your page right now, you'll find that you can Drag n Drop rows backwards and forwards with ease. But you're not done. Just for grins, hit your "Plus" button for a form row, then try to drag it. It's not easy to grab it, but it can be done. Suddenly you're trying to re-order stuff that isn't ready for re-order!

This is easily bypassed, by adding some configuration to your sortable. The jQueryUI Sortable has it's own **API and configuration options**, and you can easily apply them to your Angular UI Sortable as well. First, create a configuration object in your controller. We don't need much, just something that identifies what our drag handles are (if there's no handle, it can't be dragged):

```
$scope.sortConfig = {

 handle: ".well"

};
```

We aren't applying the *well* class to edit rows, so this will exclude them. All we have to do is tell our directive to use this config:

```
<div ui-sortable="sortConfig" ng-model="bands" class="container-fluid">
```

Now, if you reload and try it again, you'll see that you can no longer Drag n Drop the edit rows.

But, as you continue to drag items around, you'll notice that your display order column of data is way out of wack after a bit. The order should still be "1, 2, 3, etc", but now it's just a jumble. You know that the output is tied to your model, by accessing the *band.rank*, so as items get dragged around the *rank* needs to get updated. This can easily be done by *watch*ing the *bands* variable in the controller. We can set a method that says "when this changes, update it":

```
$scope.$watch(
 "bands",
 function ( newValue, oldValue ) {

  for ( var i = 0; i < newValue.length; i++ ) {

   newValue[i].rank = i+1;

  }

 },
 true
);
```

The *$watch* function allows you to set model listeners. Here we are looking for any change to the *bands* array in the *$scope*. Since *bands* is a complex data type we include the "true" to match objectEquality, meaning it will watch it "deeply", for any change, and fire when it changes. We're looping every item in the array, and resetting the rank according to it's new position in the array. The *rank* may never have changed (each time a new row is added this will also fire), but the overhead is pretty minimal, and it's a simple way to ensure the display order is always right.

That's it. Sample Code can be pulled from the download link. If you have questions, comments, or war stories, just hit me with a comment below.