**Exercise05\_01\_01 – Step 1**



1. Copy the ***Exercise05\_01\_01.zip*** file from Google Classroom to your Desktop. Unzip the files to create a project folder in your working sandbox structure. Open the new project with your IDE. Let’s take a look at the ***index.html*** file. It is a pretty simple form with just four ***<input>*** fields. We will concentrate on that ***<fieldset>*** for validation.
2. Our first task is to link in the AngularJS ***libraries***. They have already been downloaded from the AngularJS.org website, and are located in the ***js/lib/angular*** folder in the project. Because AngularJS scripts should always be placed at the top of the HTML file, let’s do that just above the closing ***</head>*** tag as follows:  
    ***<script src="js/lib/angular/angular.min.js"></script>***</head>  
   Test this in the browser with a Live Preview, with Developer Tools on, and in the Console tab, to make sure there are no errors.
3. We must specify which part of our application is going to be under AngularJS control. For this we will use an ***ng-app*** directive. We can limit this for this app to the area that is holding our ***<form>***, which is the ***<section>*** element surrounding it:  
    ***<section class="card register" ng-app>***
4. We will now implement our data binding abilities. To do this, let’s specify that our first input field is a ***Model***. We will do this with an ***ng-model*** directive and name the model ***message*** for now:  
    ***<input type="text" name="firstname" placeholder="First Name"   
    ng-model="message">***
5. To test the data binding, we will need an AngularJS ***expression*** that test the Model. At the bottom of the first ***<div>*** in the ***<form>***, let’s add a ***<p>*** with an expression as follows:  
    <div class="textintro">  
    <h1>Form Validation</h1>  
    <p>Please fill out the following form</p>  
    ***<p>{{ message }}</p>*** </div>  
   Let’s test this in the browser by regenerating and putting some data in the first ***<input>*** field.

**Exercise05\_01\_01 – Step 2**



1. To start to implement our ***validation***. Let’s first modify our test ***data binding***, to something more useful. In the ***index.html*** file, we can modify the ng-model we set, and add directives to the other input fields:  
    <input type="text" name="firstname"   
    placeholder="First Name"   
    ***ng-model="firstname"***>  
    <input type="text" name="lastname"   
    placeholder="Last Name"  
    ***ng-model="lastname"***>  
    <input type="email" name="email"   
    placeholder="Email"  
    ***ng-model="email"***>  
    <input type="password" name="password"   
    placeholder="Password"  
    ***ng-model="password"***>
2. We can now fix our AngularJS expression that is bound to the first <input> field as it no longer works:  
   ***<p>{{ firstname }}</p>***Let’s test this in the browser by regenerating and putting some data in the first ***<input>*** field.
3. We can get a little bit more sophisticated by placing all of these ***ng-model*** values into a single ***object*** named ***user*** as follows:  
    ***ng-model="user.firstname"***>  
    ***ng-model=" user.lastname"***>  
    ***ng-model=" user.email"***>  
    ***ng-model=" user.password"***>
4. Now fix our AngularJS expression and bind it to the entire object:  
   ***<p>{{ user }}</p>***Let’s test this in the browser by regenerating and putting some data in the various ***<input>*** fields. Unfortunately, for the minute, it is showing our password data in the clear. Also, if we put in an invalid ***email*** and ***Submit***, we get a standard HTML validation error.
5. We should now turn off the standard HTML validation, which we can do with a single attribute in the ***<form>*** element  
    ***<form name="myform" novalidate>***Let’s give this a test in the browser and we should no longer get error messages on a bad email Submit.
6. We can begin to make use of the AngularJS directives at this point. We will start by making the first <input> field required:  
    <input type="text" name="firstname"   
    placeholder="First Name"   
    ng-model="user.firstname"  
    ***ng-required=”true”***>
7. Now let’s demonstrate the use of an injected ***properties***. We can change the ***<p>*** element that we used for testing to use a ***property*** as follows:  
    ***<p ng-show="myform.firstname.$invalid">  
    You must fill out your first name.</p>***Let’s give this a test in the browser and we should see the message at the top of the screen. It goes away when we type data in the First Name field.
8. Let’s examine the behavior of the injected ***classes***. Regenerate the browser and open Developer Tools to the Elements tab. Expand until the first ***<input>*** field is visible. Notice the classes which have been injected. One of them is ***ng-invalid***, among others. Also look at the <form> element itself, which contains an ***ng-invalid*** class. Type data into the field and you can see the classes change, in particular, we now have ***ng-valid*** showing in both the ***<form>*** and ***<input>*** elements.

**Exercise05\_01\_01 – Step 3**



1. Let’s go deeper into using AngularJS validation rules by improving the form validation. We can start by moving our ***<p>*** element with the required message to directly below the ***<input>*** field to which it refers:  
    ng-required="true">  
    ***<p ng-show="myform.firstname.$invalid">  
    You must fill out your first name.</p>***Let’s test this in the browser and the message is in a more appropriate place. It disappears when data is entered.
2. This is working so far as it goes, but it is annoying that the message is on when the form first appears. We can fix that behavior by using ***multiple*** ***properties*** in combination. We will only turn on the message if the field has been entered and left blank by using the ***$touched*** property:  
    <p ng-show="myform.firstname.$invalid ***&&  
    myform.firstname.$touched"***>  
   Let’s test this in the browser, enter and leave the field, and the message appears. Regenerate and never enter the field, and it will not come on. We will improve this more later.
3. Now let’s copy those changes, with some minor modifications, to the ***Last*** ***Name*** field:  
    ng-model="user.lastname" ***ng-required="true">  
    <p ng-show="myform.lastname.$invalid &&  
    myform.lastname.$touched">  
    You must fill out your last name.</p>***Let’s test this in the browser.
4. Let’s build in some support for later visuals on our errors. Let’s put in a ***class*** in each of our message paragraphs. This is so we can hook it for some ***CSS*** later. Add the following class to each of the two ***Name*** <input> fields:  
    ***<p class="error validationerror"*** ng-show="myform.firstname.$invalid &&  
     
    ***<p class="error validationerror"*** ng-show="myform.lastname.$invalid &&
5. Let’s do some work on our ***Email*** field, which will not be ***required***. However, AngularJS is smart enough to know that it’s ***type*** is ***email***. Let’s copy down one of our error message ***<p>*** elements to directly below the email ***<input>*** and make some necessary syntactic modifications:  
    ng-model="user.email">  
    ***<p class="error validationerror"   
    ng-show="myform.email.$invalid &&  
    myform.email.$touched">  
    This must be a valid email.</p>***Let’s test this in the browser, enter and leave the field, and the message does not appear, it is not required. Enter an invalid email, then try a valid one. As we can see, AngularJS is smart enough to test email validity without ***code*** or ***Regex***.
6. Let’s do a little different validation with the Password field, and test for a minimum length:  
    ng-model="user.password"  
    ***ng-minlength="6">  
    <p class="error validationerror"   
    ng-show="myform.password.$invalid &&  
    myform.password.$touched">  
    Password must be at least 6 characters.</p>***  
   Let’s test this in the browser for minimum length.

**Exercise05\_01\_01 – Step 4**



1. We can use the amazing validation powers of AngularJS to power styling for us in the process. To set the stage, we will modify our ***CSS*** for some error support. Open up ***style.css*** from the ***/css*** folder. We can use the classes that have been injected into our <input> fields to get some style. Scroll all the way to the bottom and add the following style rule to take care of our invalid fields:  
   ***input.ng-invalid {  
    border: 1px solid #DA3637;  
   }***Let’s test this in the browser to see the effects. The ***Name*** fields will start off red-bordered. The others will turn red only as long as data is invalid. Virtually no code!
2. Let’s fix the initial red, much as we did with the error messages. We can still use the CSS style rule with a modification:  
   ***input.ng-invalid.ng-touched {***Let’s test this in the browser, and it is now working in conjunction with the error messages.
3. Let’s build in some support for positive user feedback. Copy the previous CSS style rule down below itself, and we will do some modifications as follows:  
   ***input.ng-valid.ng-touched {  
    border: 1px solid #509D12;  
   }***Let’s test this in the browser to see the effects.
4. We can do some work on the **Submit** button to let the user know when the ***<form>*** is not completely valid by graying out the button. For this we can use the ***classes*** in the **<form>** element itself and arrange some CSS to do the job. Let’s build another style rule at the bottom:  
   ***.ng-invalid .btn {  
    background-color: gray;  
   }***Let’s test this in the browser with all of our fields.
5. We should also do a little bit of highlight on our ***error*** messages, and also get a little bit more room for the messages, which will just take a little more CSS at the bottom:  
   ***.error {  
    color: red;  
    padding-bottom: 10px;  
   }***Let’s test this in the browser for effect.

**Exercise05\_01\_01 – Step 5**



1. To implement form submission, we will first need a JavaScript file. In the ***/js*** folder, let’s create a file named ***app.js***. We need to link that into index.html, below the AngularJS library:  
    <script src="js/lib/angular/angular.min.js"></script>  
    ***<script src="js/app.js"></script>***
2. Now we can create an app ***module*** in ***app.js***. It will have no dependencies:  
   ***var myApp = angular.module('myApp', []);***Let’s test this in the browser, and it is now working in conjunction with the error messages.
3. We need to go back to the ***index.html*** file and give a value to the ***ng-app*** directive that has the new module name:  
   ***<section class="card register" ng-app="myApp">***Let’s test this in the browser to see everything still works.
4. We need to go back to app.js and scaffold out a Controller for this application. We will ***name*** it and give it an ***array*** of its ***dependencies***:  
   ***myApp.controller('RegistrationController', [’$scope’, function($scope) {  
      
   }]);***
5. Let’s do something different, and add a ***method*** called ***register*** to our ***$scope*** variable that will handle our ***<form>*** submission. It will use our ***$scope variable***. and ***two way data binding*** to do this:  
   myApp.controller('RegistrationController', [$scope, function($scope) {  
    ***$scope.register = function() {  
    $scope.message = 'Thank you for your submission, ' +   
    $scope.user.firstname;  
    };***}]);
6. We need to go to ***index.html*** and build the code that will use our new message. We can add a new directive to our ***<form>*** tag, ***ng-submit***, and set its value to the function that we built, because it is in our app scope:  
    <form name="myform"   
    ***ng-submit="register()"*** novalidate>
7. Now we just need to set up a ***<p>*** for our message, with content that is an AngularJS ***expression*** that is bound to it:  
    <p>Please fill out the following form</p>  
    ***<p ng-show="message">{{ message }}</p>***
8. Lastly we need to use an ng-controller directive to tie our new Controller into the area it is managing as follows:  
    <section class="card register"   
    ng-app="myApp"  
    ***ng-controller="RegistrationController"***>  
   Let’s give the app a full browser test with a submit.