Agenda: Forms

- Using Simple Form
- Working with Select and Options
- Input Validations
- Using CSS classes
- Form Events
- Custom Model update triggers
- Custom Validations

Working with Simple form

The most important directive used in form is ng-model. Its used for two-way data binding by synchronizing view and model.

```
<div ng-app="myApp" ng-controller="studentController">
   <form>
      Name: <input type="text" ng-model="user.name" /><br />
      E-mail: <input type="email" ng-model="user.email" /><br />
      Gender:
      <input type="radio" ng-model="user.gender" value="male" />Male
      <input type="radio" ng-model="user.gender" value="female" />Female<br />
      <input type="button" ng-click="reset()" value="Reset" />
      <input type="submit" ng-click="update()" value="Save" />
   </form>
   <pre>form = {{user | json}}</pre>
   <pre>master = {{master | json}}</pre>
</div>
<script>
   var myApp = angular.module('myApp', []);
   myApp.controller('studentController', ['$scope', function ($scope) {
      $scope.user = new function () {
         this.name = "Sandeep"
         this.email = "sandeepsoni@deccansoft.com"
         this.gender = "male"
      }
      $scope.master = {};
      $scope.update = function () {
         $scope.master = angular.copy($scope.user);
```

```
    };
    $scope.reset = function () {
        $scope.user = angular.copy($scope.master);
    };
  }]);
</script>
```

The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation.

Working with Select and Options

ngOptions: in one of the following forms:

- for array data sources:
  - *label* **for** *value* **in** *array*
  - *value* ***as*** *label* **for** *ele* **in** *array*
  - *label* **group by** *group* **for** *value* **in** *array*
- for object data sources:
  - label for (key , value) in object
  - label group by group for (key, value) in object

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
  <script src="angular.js"></script>
</head>
<body>
  <div ng-app="myApp" ng-controller="studentController">
    Name: <input type="text" name="txtName" ng-model="nameFilter" /> <br />
    Subject(null not allowed):
    <select ng-options="sub.Name for sub in subjects" id="s1" ng-model="selectedSubject"></select> <br />
    <input type="button" onclick="alert(document.getElementById('s1').value);" /> {{selectedSubject}}
    <hr />
   <select ng-options="sub.Id as sub.Name for sub in subjects" id="s1" ng-model="selectedSubject"></select>
    <br />
    Subject(null allowed)::
    <select ng-options="sub.Name for sub in subjects" ng-model="selectedSubject">
      <option value="">All</option>
```

```
    </select><br />

    <hr />
    Subject(Grouped): <select ng-options="sub.Name group by sub.Level for sub in subjects"
                      ng-model="selectedSubject"></select>
  </div>

  <script>
    var myApp = angular.module('myApp', []);
    myApp.controller('studentController', ['$scope', function ($scope) {
      $scope.subjects = [{ Id: 11, Name: "Maths", Level: 'Basic' },
                         { Id: 21, Name: "Science", Level: 'Intermediate' },
                         { Id: 31, Name: "Social", Level: 'Basic' }];
      $scope.selectedSubject = 11;// $scope.subjects[0];
    }]);
  </script>
</body>
</html>
```

Form States and Methods

Form properties / state

- **$pristine**: True if user has not interacted with the form yet
- **$dirty**: True if user has already interacted with the form
- **$valid**: True if all of the containing forms and controls are valid
- **$invalid**: True if at least one containing control or form is invalid
- **$submitted**: True if user has submitted the form even if its invalid.
- **$error**: Is an object hash, containing references to controls or forms with failing validators

Control Properties (Not valid for Form):

- **$touched:** True if control has lost focus.
- **$untouched:** True if control has not lost focus yet

Form Methods:

- **$setDirty():** Sets the form to a dirty state.

This method can be called to add the 'ng-dirty' class and set the form to a dirty state. This method will also propagate to parent forms.

- **$setPristine():** Sets the form to its pristine state.
  This method can be called to remove the 'ng-dirty' class and set the form to its pristine state (ng-pristine class). **This method will also propagate to all the controls contained in this form**.
  Setting a **form** back to a pristine state is often useful when we want to 'reuse' a form after saving or resetting it.
- **$setUntouched():** Sets the form elements to their untouched state.
  This method can be called to remove the 'ng-touched' class and set the form **controls** to their untouched state (ng-untouched class).
  Setting a **form controls** back to their untouched state is often useful when setting the form back to its pristine state.
- **$setSubmitted():** Sets the form to its submitted state.

Using CSS classes

To allow styling of form as well as controls, ngModel adds these CSS classes:

- ng-valid: the model is valid
- ng-invalid: the model is invalid
- ng-valid-[key]: for each valid key added by $setValidity
- ng-invalid-[key]: for each invalid key added by $setValidity
- ng-pristine: the control hasn't been interacted with yet
- ng-dirty: the control has been interacted with
- ng-touched: the control has been blurred
- ng-untouched: the control hasn't been blurred.

```
<style>
  form input.ng-untouched {
    background-color: yellow;
  }
  form input.ng-invalid.ng-touched {
    background-color: red;
  }
  form input.ng-valid.ng-touched {
    background-color: green;
  }
</style>
```

Validating Input Elements

- **$error**: Is an object hash, containing references to controls or forms with failing validators, where:
  - keys are validation tokens (error names),
  - values are arrays of controls or forms that have a failing validator for given error name.

    Built-in error validation tokens:

    - email
    - max
    - maxlength
    - min
    - minlength
    - number
    - pattern
    - required
    - url
    - date
    - datetimelocal
    - time
    - week
    - month

    ```
    <pre>Errors: {{user_form.$error | json}}</pre>
    ```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
  <script src="angular.js"></script>
</head>
<body>
  <div ng-app="myApp" ng-controller="studentController">
    <form novalidate class="simple-form" name="myForm">
      Name: <input type="text" name="username" ng-model="user.name" required />
      <span style="color:red" ng-show="myForm.username.$dirty && myForm.username.$invalid">
        <span ng-show="myForm.username.$error.required">Username is required.</span>
      </span>
      <br />
      E-mail: <input type="email" name="useremail" ng-model="user.email" required />
      <span style="color:red" ng-show="myForm.useremail.$dirty && myForm.useremail.$invalid">
        <span ng-show="myForm.useremail.$error.required">Email is required.</span>
        <span ng-show="myForm.useremail.$error.email">Please provide valid email.</span>
      </span><br />
      <p>
        <input type="submit" ng-disabled="myForm.$invalid"> <br />
        <input type="button" ng-click="reset(myForm)" value="Reset" />
        <input type="submit" ng-click="update(user)" value="Save" />
      </p>
    </form>
   <pre>Errors: {{myForm.$error | json}}</pre>
  </div>
  <script>
    var myApp = angular.module('myApp', []);
    myApp.controller('studentController', ['$scope', function ($scope) {
      $scope.user = new function () {
        this.name = ""
        this.email = ""
      }
      $scope.master = {};
```

```
        $scope.update = function (user) {
            $scope.master = angular.copy(user);
        };
        $scope.reset = function (form) {
            if (form) {
                form.$setPristine();
                form.$setUntouched();
            }
            $scope.user = angular.copy($scope.master);
        };
        $scope.reset();
    }]);
  </script>
</body>
</html>
```

Validating Form elements in a repeater

ngForm directive: The ng-form directive allows for nesting forms that can be used for things like partial validation. The idea is simple, on each repeated element, add an ng-form directive with a name. Then inside that, you can now reference inputs by name on that subform.

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title></title>
  <script src="angular.js"></script>
</head>
<body ng-app="myApp" ng-controller="validationDemo">
  <form action="/" name="myForm" method="post">
    <table>
      <tr ng-repeat="item in Items" ng-form="subform">
        <td>
          <input type="text" name="txtKey" ng-model="item.key" required />
          <span ng-show="subform.txtKey.$error.required">Required</span>
        </td>
```

```
        </tr>
      </table>
      <hr />
    </form>
    <script>
      var myApp = angular.module("myApp", [])
      myApp.controller("validationDemo", function ($scope) {
        $scope.Items = [{ key: "V1" }, { key: "V2" }, { key: "V3" }, { key: "V4" }]
      })
    </script>
</body>
</html>
```

Model Update Options

By default, any change to the content will trigger a model update and form validation. You can override this behavior using the ngModelOptions directive to bind only to specified list of events.

ng-model-options="{ updateOn: 'blur' }" will update and validate only after the control loses focus.

You can set several events using a **space delimited** list i.e**. ng-model-options**="{ **updateOn**: **'mousedown blur'** }"

Model Options Properties:

- **updateOn**: string specifying which event should the input be bound to. You can set several events using an space delimited list. There is a special event called default that matches the default events belonging of the control.

  ng-model-options="{ updateOn: 'blur' }"

- **debounce**: integer value which contains the debounce model update value in milliseconds. A value of 0 triggers an immediate update. If an object is supplied instead, you can specify a custom value for each event

  ng-model-options="{ debounce: {'default': 500, 'blur': 0} }"

- **getterSetter**: boolean value which determines whether or not to treat functions bound to ngModel as getters/setters.

```
var privateValue = 'Testing';
$scope.user = {
   firstName: getSetFirstName, // <-- remember, this is what our input's ng-model is bound to
};
// Here's the getter/setter function
```

```
function getSetFirstName(newValue) {
   // if there is no newValue specified then it's being invoked as a getter But this will be a problem later...
   if (angular.isDefined(newValue)) {
      privateValue = newValue.toUpperCase();
   }
   return privateValue;
}
$scope.getPrivateValue = function () {
   alert(privateValue);
}
<input type="text" name="name" ng-model="user.firstName"
      ng-model-options="{ getterSetter: true}" />
<input type="button" name="btn" value="Get" ng-click="getPrivateValue()" />
```

- **allowInvalid**: boolean value which indicates that the model can be set with values that did not validate correctly instead of the default behavior of setting the model to undefined.

```
<input type="text" name="name" ng-model="user.firstName"
ng-model-options="{ getterSetter: true, allowInvalid: true}" required />
```

- **timezone**: Defines the timezone to be used to read/write the Date instance in the model for <input type="date">, <input type="time">, ... It understands UTC/GMT and the continental US time zone abbreviations, but for general use, use a time zone offset, for example, '+0430' (4 hours, 30 minutes east of the Greenwich meridian) If not specified, the timezone of the browser will be used.

```
Time: <input type="time" name="txtDate" ng-model="timeDemo"
ng-model-options="{timezone:'UTC'}" />
```

Complete Examples:

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="utf-8" />
   <title></title>
   <script src="angular.js"></script>
</head>
<body ng-app="myApp" ng-controller="demoController">
   <form action="/" name="myForm" method="post">
      First Name: <input type="text" name="txtFirstName" ng-model="user.firstName" required
```

```
    ng-model-options="{updateOn: 'default blur mouseover',
                       debounce: {'default': 5000, 'blur': 0} ,
                       getterSetter: true,
                       allowInvalid: true }" /> <br />
    Last Name: <input type="text" name="txtLastName" ng-model="user.lastName" /> <br />
    <input type="button" name="btnFullName" value="GetFullName" ng-click="fullName()" />
    <input type="time" name="txtDemo" ng-model="demoTime" ng-model-options="{timezone: 'GMT'}" />
{{demoTime}}
  </form>
  <pre>{{user | json}} </pre>
  <script>
    var myApp = angular.module("myApp", [])
    myApp.controller('demoController', function ($scope) {
      $scope.user = { firstName: getSetFirstNameValue, lastName: 'Soni' }
      $scope.fullName = function () {
        alert(priFirstName + " " + $scope.user.lastName)
      }
      var priFirstName = 'SANDEEP'
      function getSetFirstNameValue(newValue)
      {
        if (angular.isDefined(newValue))
          priFirstName = newValue.toUpperCase();
        return priFirstName;
      }

    })
  </script>
</body>
</html>
```

Custom Validations

- Angular provides basic implementation for most common HTML5 input types: (text, number, url, email, date, radio, checkbox), as well as some directives for validation (required, pattern, minlength, maxlength, min, max).
- For custom validation we have to do is build a directive that requires ngModel. Requiring ngModel will pass the ngModelController into the linking function as the fourth argument. The ngModel controller has a lot of

handy functions on it, in particular there is **$setValidity**, which allows you to set the state of a model field has $valid or $invalid as well as set an $error flag.

- We can add a validation functions to the **$validators** object on the ngModelController. Each function in the $**validators** object receives the **modelValue** and the **viewValue** as parameters. Angular will then call **$setValidity** internally with the function's return value (true: valid, false: invalid).
- The validation functions are executed every time an input is changed ($setViewValue is called) or whenever the bound model changes.

To get a hold of the controller, you require it in the directive as shown in the example below.

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title></title>
  <script src="angular.js"></script>
  <script src="jquery-2.1.1.js"></script>
</head>
<body ng-app="myApp" ng-controller="validationController">
  <form action="/" method="post" name="myForm">
    <input type="email" name="txtEmail" ng-model="email" />
    <span ng-show="myForm.txtEmail.$error.email">Please enter valid email format</span>
    <br />
    Model Value: {{myForm.txtEmail.$modelValue}} <br />
    View Value: {{myForm.txtEmail.$viewValue}}
    <hr />
    <input type="text" style="color:blue" name="txtNumber" ng-model="number1" integer />
    <span ng-show="myForm.txtNumber.$error.integer">Please enter valid Integer Number</span>
    <br />
    Model Value: {{myForm.txtNumber.$modelValue}} <br />
    View Value: {{myForm.txtNumber.$viewValue}}
    <hr />
  </form>
  <script>
    var myApp = angular.module("myApp", [])
    myApp.controller("validationController", function ($scope) {
```

```
    })
    myApp.directive("integer", function () {
      return {
        require: 'ngModel',
        link: function(scope, elem, attrs, ctrl)
        {
          var originalColor = $(elem).css('color')

          ctrl.$validators.integer = function(modelValue, viewValue)
          {
            if (ctrl.$isEmpty(modelValue))
              return true;
            if (isNaN(viewValue) || viewValue.toString().indexOf('.') != -1)
            {
              $(elem).css('color','red')
              return false;
            }
            $(elem).css('color', originalColor)
            return true;
          }
        }
      }
    })
  </script>
</body>
</html>
```