Angular JS Notes

www.enosislearning.com

Contents

[What is AngularJS? 4](#_Toc518980401)

[Features 4](#_Toc518980402)

[Core Features 5](#_Toc518980403)

[Concepts 5](#_Toc518980404)

[Advantages of AngularJS 6](#_Toc518980405)

[Disadvantages of AngularJS 6](#_Toc518980406)

[**The Model** 7](#_Toc518980407)

[**The View** 7](#_Toc518980408)

[**The Controller** 8](#_Toc518980409)

[How AngularJS integrates with HTML 8](#_Toc518980410)

[Example1 8](#_Toc518980411)

[Steps to create AngularJS Application 8](#_Toc518980412)

[Step 1: Load framework 8](#_Toc518980413)

[Step 2: Define AngularJS Application using ng-app directive 8](#_Toc518980414)

[Step 3: Define a model name using ng-model directive 8](#_Toc518980415)

[Step 3: Bind the value of above model defined using ng-bind directive. 8](#_Toc518980416)

[Steps to run AngularJS Application 8](#_Toc518980417)

[Output : Enter your name and see the result. 9](#_Toc518980418)

[How AngularJS integrates with HTML 9](#_Toc518980419)

[**ng-app directive :**ng-app directive starts an AngularJS Application. It defines the root element 9](#_Toc518980420)

[**ng-init directive** 9](#_Toc518980421)

[**ng-model directive** 10](#_Toc518980422)

[**ng-repeat directive** 10](#_Toc518980423)

[**Output :**Open textAngularJS.htm in a web browser. Enter your name and see the result. 11](#_Toc518980424)

[**Using numbers :** <p>Expense on Books : {{cost \* quantity}} Rs</p> 11](#_Toc518980425)

[**Using strings :**<p>Hello {{student.firstname + " " + student.lastname}}!</p> 11](#_Toc518980426)

[**Using object :** <p>Roll No: {{student.rollno}}</p> 11](#_Toc518980427)

[**Using array :** <p>Marks(Math): {{marks[3]}}</p> 11](#_Toc518980428)

[**Example** 11](#_Toc518980429)

[Views 12](#_Toc518980430)

[Controller 12](#_Toc518980431)

[Scope / View Model 12](#_Toc518980432)

[Scope Inheritance 13](#_Toc518980433)

[Example 14](#_Toc518980434)

[Result 14](#_Toc518980435)

[AngularJS Controllers 15](#_Toc518980436)

[AngularJS Example 15](#_Toc518980437)

[Controller Methods 16](#_Toc518980438)

[AngularJS Example 16](#_Toc518980439)

[Controllers In External Files 16](#_Toc518980440)

[AngularJS Example 16](#_Toc518980441)

[Another Example 17](#_Toc518980442)

[uppercase filter 21](#_Toc518980443)

[lowercase filter 21](#_Toc518980444)

[ng-disabled directive 22](#_Toc518980445)

[ng-show directive 22](#_Toc518980446)

[ng-hide directive 22](#_Toc518980447)

[ng-click directive 22](#_Toc518980448)

[Example 22](#_Toc518980449)

[Events 24](#_Toc518980450)

[ng-click 24](#_Toc518980451)

[Validate data 24](#_Toc518980452)

[Example 25](#_Toc518980453)

[Required 27](#_Toc518980454)

[E-mail 27](#_Toc518980455)

[Form State and Input State 27](#_Toc518980456)

[Example 28](#_Toc518980457)

[CSS Classes 28](#_Toc518980458)

[Example 29](#_Toc518980459)

[Example : Apply styles for unmodified (pristine) forms, and for modified forms: 29](#_Toc518980460)

[Custom Validation 29](#_Toc518980461)

[Example 29](#_Toc518980462)

[Example Explained: 30](#_Toc518980463)

[Validation Example 31](#_Toc518980464)

[Enter ngMessages 33](#_Toc518980465)

[Required Fields 33](#_Toc518980466)

[Email 33](#_Toc518980467)

[Regular Expressions 34](#_Toc518980468)

[Minlength and Maxlength 34](#_Toc518980469)

[A Final Tweak 35](#_Toc518980470)

# What is AngularJS?

AngularJS is an open source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.4.3.

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly.

Angular's data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology.

### Features

* AngularJS is a powerful JavaScript based development framework to create RICH Internet Application(RIA).
* AngularJS provides developers options to write client side application (using JavaScript) in a clean MVC(Model View Controller) way.
* Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
* AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

Overall, AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain.

### Core Features

Following are most important core features of AngularJS −

* **Data-binding** − It is the automatic synchronization of data between model and view components.
* **Scope** − These are objects that refer to the model. They act as a glue between controller and view.
* **Controller** − These are JavaScript functions that are bound to a particular scope.
* **Services** − AngularJS come with several built-in services for example $https: to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
* **Filters** − These select a subset of items from an array and returns a new array.
* **Directives** − Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ngBind, ngModel...)
* **Templates** − These are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".
* **Routing** − It is concept of switching views.
* **Model View Whatever** − MVC is a design pattern for dividing an application into different parts (called Model, View and Controller), each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.
* **Deep Linking** − Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.
* **Dependency Injection** − AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

### Advantages of AngularJS

* AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
* AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience
* AngularJS code is unit testable.
* AngularJS uses dependency injection and make use of separation of concerns.
* AngularJS provides reusable components.
* With AngularJS, developer write less code and get more functionality.
* In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

On top of everything, AngularJS applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.

### Disadvantages of AngularJS

* **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
* **Not degradable** − If your application user disables JavaScript then user will just see the basic page and nothing more.

**M**odel **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

* Model - It is the lowest level of the pattern responsible for maintaining data.
* View - It is responsible for displaying all or a portion of the data to the user.
* Controller - It is a software Code that controls the interactions between the Model and View.

MVC is popular because it isolates the application logic from the user interface layer and supports separation of concerns. The controller receives all requests for the application and then works with the model to prepare any data needed by the view. The view then uses the data prepared by the controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.



**The Model**

The model is responsible for managing application data. It responds to the request from view and to the instructions from controller to update itself.

**The View**

A presentation of data in a particular format, triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.

**The Controller**

The controller responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.

AngularJS is a MVC based framework. AngularJS uses MVC methodology.

Before we start with creating actual HelloWorld application using AngularJS, let us see what are the actual parts of aAngularJS application. An AngularJS application consists of following important parts.

## How AngularJS integrates with HTML

* ng-app directive indicates the start of AngularJS application.
* ng-model directive then creates a model which can be used with the html page and within the div having ng-app directive.
* ng-bind then uses the name model to be displayed in the html span tag whenever user input something in the text box.
* Closing</div> tag indicates the end of AngularJS application
* ng-repeat - This directive repeats html elements for each item in a collection

**ng-app directive :** ng-app directive starts an AngularJS Application. It defines the root element

<div ng-app="">

...

</div>

**ng-init directive:** ng-init directive initializes an AngularJS Application data.

It is used to put values to the variables to be used in the application. In following example, we'll initialize an array of countries. We're using JSON syntax to define array of countries.

<div ng-app="" ng-init="countries=[{locale:'en-US',name:'United States'},

 {locale:'en-GB',name:'United Kingdom'},

 {locale:'en-FR',name:'France'}]">

...

</div>

**ng-model directive:**ng-model directive defines the model/variable to be used in AngularJS Application. In following example, we've defined a model named "name".

<div ng-app="">

<p>Enter your Name: <input type="text" ng-model="name"></p>

</div>

**ng-repeat directive:**ng-repeat directive repeats html elements for each item in a collection.. In following example, we've iterated over array of countries.

<div ng-app="">

...

<p>List of Countries with locale:</p>

<ol>

<li ng-repeat="country in countries">

 {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

</li>

</ol>

</div>

Output: Enter your name and see the result.



|  |
| --- |
| Steps to create AngularJS ApplicationStep 1: Load frameworkBeing a pure JavaScript framework, it can be added using <Script> tag.<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>Step 2: Define AngularJS Application using ng-app directive<div ng-app="">...</div>Step 3: Define a model name using ng-model directive<p>Enter your Name: <input type="text" ng-model="name"></p>Step 3: Bind the value of above model defined using ng-bind directive.<p>Hello <span ng-bind="name"></span>!</p>Steps to run AngularJS ApplicationUse above mentioned three stpes in an HTML page.*testAngularJS.htm*<html><title>AngularJS First Application</title><body><h1>Sample Application</h1><divng-app=""><p>Enter your Name: <inputtype="text"**ng-model="name"**></p><p>Hello <span **ng-bind="name"**></span>!</p></div><scriptsrc="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script></body></html> |

***Example: testAngularJS.htm***

<html>

<title>AngularJS Directives</title>

<body>

<h1>Sample Application</h1>

<div ng-app="" ng-init="countries=[{locale:'en-US',name:'United States'},

 {locale:'en-GB',name:'United Kingdom'},

 {locale:'en-FR',name:'France'}]">

<p>Enter your Name: <input type="text" ng-model="name"></p>

<p>Hello <span ng-bind="name"></span>!</p>

<p>List of Countries with locale:</p>

<ol>

<li ng-repeat="country in countries">

 {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

</li>

</ol>

</div>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

</body>

</html>

**Output :**Open textAngularJS.htm in a web browser. Enter your name and see the result.



data where they are used.

**Using numbers :** <p>Expense on Books : {{cost \* quantity}} Rs</p>

**Using strings :**<p>Hello {{student.firstname + " " + student.lastname}}!</p>

**Using object :** <p>Roll No: {{student.rollno}}</p>

**Using array :** <p>Marks(Math): {{marks[3]}}</p>

**Example**

<html>

<title>AngularJS Expressions</title>

<body>

<h1>Sample Application</h1>

<div ng-app="" ng-init="quantity=1;cost=30; student={firstname:'Mahesh',lastname:'Parashar',rollno:101};marks=[80,90,75,73,60]">

<p>Hello {{student.firstname + " " + student.lastname}}!</p>

<p>Expense on Books : {{cost \* quantity}} Rs</p>

<p>Roll No: {{student.rollno}}</p>

<p>Marks(Math): {{marks[3]}}</p>

</div>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

</body>

</html>

We are going to see Models Views and Controllers and will see model part of the application also called View Model or $scope in details. It is a really important part of AngularJs.
**View Controller and Scope**



### Views

The AngularJS application has a View which is the part rendered in a browser, it is the way to show the data to users. View uses the directives, filters and data-bindings. But to make view simple and easy to maintainable we do not put all of our code into these View. Separating code from views also make it easy to write tests for the business logic.

**Controller**

We put all of our logic to container called controller in Angular. The Controller controls and prepare the data into the form so it can be rendered at the View. So Controller actually transforms the bunch of data into the representational form and also take from view and set into the Model after validating it. The controller is responsible for communicating the server code to fetches the data from a server using Ajax calls and send the data to back-end server from Views.



### Scope / View Model

The most important part of the architecture is $scope or Model or View Model. It is the actual link between Controllers and Views. There can be a controller which we can bind to two or more views. Like controller for a registration form can have a different view for desktop and another view for mobile. In real Controller has no information about the Views and similarly View is independent of logic implemented or data present in the Controller. $scope acts as the communication tunnel between the Views and Controller.

Scope is a special javascript object which plays the role of joining controller with the views. Scope contains the model data. In controllers, model data is accessed via $scope object.

<script>

Var mainApp=angular.module("mainApp",[]);

mainApp.controller("shapeController",function($scope){

 $scope.message="In shape controller";

 $scope.type="Shape";

Alert($scope.type);

});

</script>

**Following are the important points to be considered in above example.**

* $scope is passed as first argument to controller during its constructor definition.
* $scope.message and $scope.type are the models which are to be used in the HTML page.
* We've set values to models which will be reflected in the application module whose controller is shapeController.
* We can define functions as well in $scope.

**Scope Inheritance**

Scope are controllers specific. If we defines nested controllers then child controller will inherit the scope of its parent controller.

<script>

Var mainApp=angular.module("mainApp",[]);

mainApp.controller("shapeController",function($scope){

 $scope.message="In shape controller";

 $scope.type="Shape";

});

mainApp.controller("circleController",function($scope){

 $scope.message="In circle controller";

});

</script>

Following are the important points to be considered in above example.

* We've set values to models in shapeController.
* We've overridden message in child controller circleController. When "message" is used within module of controller circleController, the overridden message will be used.

**Example**

Following example will showcase all the above mentioned directives.

testAngularJS.htm

<html>

<head>

<title>Angular JS Forms</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app="mainApp" ng-controller="shapeController">

<p>{{message}} <br/> {{type}} </p>

<div ng-controller="circleController">

<p>{{message}} <br/> {{type}} </p>

</div>

<div ng-controller="squareController">

<p>{{message}} <br/> {{type}} </p>

</div>

</div>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

<script>

Var mainApp=angular.module("mainApp",[]);

mainApp.controller("shapeController",function($scope){

 $scope.message="In shape controller";

 $scope.type="Shape";

});

mainApp.controller("circleController",function($scope){

 $scope.message="In circle controller";

});

mainApp.controller("squareController",function($scope){

 $scope.message="In square controller";

 $scope.type="Square";

});

</script>

</body>

</html>

Result

Open textAngularJS.htm in a web browser. See the result.

AngularJS controllers **control the data** of AngularJS applications.

AngularJS controllers are regular **JavaScript Objects**.

**AngularJS Controllers**

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

|  |
| --- |
| **AngularJS Example:**<div ng-app="myApp" ng-controller="myCtrl">First Name: <input type="text" ng-model="firstName"><br>Last Name: <input type="text" ng-model="lastName"><br><br>Full Name: {{firstName + " " + lastName}}</div><script>var app = angular.module('myApp', []);app.controller('myCtrl', function($scope) {    $scope.firstName = "John";    $scope.lastName = "Doe";});</script> |

**Application explained:**

The AngularJS application is defined by  **ng-app="myApp"**. The application runs inside the <div>.

The **ng-controller="myCtrl"** attribute is an AngularJS directive. It defines a controller.

The **myCtrl** function is a JavaScript function.

AngularJS will invoke the controller with a **$scope** object.

In AngularJS, $scope is the application object (the owner of application variables and functions).

The controller creates two properties (variables) in the scope (**firstName** and **lastName**).

The **ng-model** directives bind the input fields to the controller properties (firstName and lastName).

##

## Controller Methods:

## The example above demonstrated a controller object with two properties: lastName and firstName.

## A controller can also have methods (variables as functions):

### AngularJS Example:

<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}
</div>
<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
</script>

## Controllers In External Files

## In larger applications, it is common to store controllers in external files.

## Just copy the code between the <script> tags into an external file named [personController.js](http://www.w3schools.com/angular/personController.js):

|  |
| --- |
| **AngularJS Example** <div ng-app="myApp" ng-controller="personCtrl">First Name: <input type="text" ng-model="firstName"><br>Last Name: <input type="text" ng-model="lastName"><br><br>Full Name: {{fullName()}}</div><script src="personController.js"></script> |

## Another Example

## For the next example we will create a new controller file:

angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
        {name:'Jani',country:'Norway'},
        {name:'Hege',country:'Sweden'},
        {name:'Kai',country:'Denmark'}
    ];
});

Save the file as  [namesController.js](http://www.w3schools.com/angular/namesController.js):

And then use the controller file in an application:

<div ng-app="myApp" ng-controller="namesCtrl">
<ul>
  <li ng-repeat="x in names">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>
</div>
<script src="namesController.js"></script>

HTML DOM

AngularJS has directives for binding application data to the attributes of HTML DOM elements.

**ng-show Directive :** To make visible true or false

**Examples:**

|  |
| --- |
| <!DOCTYPE html><html><head> <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"></script></head><body> <div ng-app=""> <p ng-show="true">Visible.</p> <p ng-show="false">Not visible.</p> </div> </body></html> |

**ng-hide Directive : To hide or unhide a control**

**Examples**

|  |
| --- |
| <!DOCTYPE html><html><head> <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"></script></head><body> <div ng-app=""> <p ng-hide="true">ng-hide="true" - Not visible.</p> <p ng-hide="false">ng-hide="false" - Visible.</p> </div> </body></html> |

**ng-disabled Directive :** The **ng-disabled** directive binds AngularJS application data to the disabled attribute of HTML elements:

**Examples**

|  |
| --- |
| <!DOCTYPE html><html><head> <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"></script></head><body> <div ng-app="" ng-init="myButton=true"> <button ng-disabled="myButton">Click Me!</button> <br /><br /> <input type="checkbox" ng-model="myButton"/>Button <br /><br /> Disabled : {{ myButton }} </div> </body></html> |

The **ng-disabled** directive binds the application data **myButton** to the HTML button's disabled attribute. The **ng-model** directive binds the value of the HTML checkbox element to the value of **myButton**. If the value of **myButton** evaluates to true, the button will be disabled.

**ng-click :** The ng-click directive defines an AngularJS click event.

**Examples**

|  |
| --- |
| <!DOCTYPE html><html><head> <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script></head><body> <div ng-app="myApp" ng-controller="countController"> <button ng-click="count = count + 1">Increment</button> {{ count }} </div> <script> var app = angular.module('myApp', []); app.controller('countController', function($scope) { $scope.count = 0; }); </script> </body></html> |

**ng-options**: Let’s change the items to an array of objects.

**Examples**

|  |
| --- |
| $scope.items = [{name: 'one', age: 30 },{ name: 'two', age: 27 },{ name: 'three', age: 50 }];and the HTML<html ng-app="app"> <body> <div ng-controller="Test"> <p>selected item is : {{selectedItem}}</p> <select ng-model="selectedItem"> <option ng-repeat="item in items" value="{{item.age}}">{{item.name}}</option> </select> </div> </body></html> |

If you notice, the selectedItem model is bound to the value of the selected option element. Here, as already explained, we’re restricted to have only strings in our items array. We cannot set the selectedItem model to the object inside items. Therefore we need to use the ng-options for these kind of situations. Let’s see how to do it.

**Re-write the above code using ng-options**:

We can re-write the above code using ng-options. The new html looks like below:

<html ng-app="app">

 <body>

 <div ng-controller="Test">

 <p>selected item is : {{selectedItem}}</p>

 <p> age of selected item is : {{selectedItem.age}} </p>

 <select ng-model="selectedItem" ng-options="item.name for item in items">

 </select>

 </div>

 </body>

</html>

Filters are used to change modify the data and can be clubbed in expression or directives using pipe character. Following is the list of commonly used filters.

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Name** | **Description** |
| 1 | Uppercase | converts a text to upper case text. |
| 2 | Lowercase | converts a text to lower case text. |

**uppercase filter**

Add uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: {{student.fullName() | uppercase}}

## lowercase filter

Add lowercase filter to an expression using pipe character. Here we've added lowercase filter to print student name in all lowercase letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Lower Case: {{student.fullName() | lowercase}}

Following directives can be used to bind application data to attributes of HTML DOM Elements.

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Name** | **Description** |
| 1 | ng-disabled | disables a given control. |
| 2 | ng-show | shows a given control. |
| 3 | ng-hide | hides a given control. |
| 4 | ng-click | represents a AngularJS click event. |

**ng-disabled directive**

Add ng-disabled attribute to a HTML button and pass it a model. Bind the model to an checkbox and see the variation.

<input type = "checkbox" ng-model = "enableDisableButton">Disable Button

<button ng-disabled = "enableDisableButton">Click Me!</button>

**ng-show directive**

Add ng-show attribute to a HTML button and pass it a model. Bind the model to an checkbox and see the variation.

<input type = "checkbox" ng-model = "showHide1">Show Button

<button ng-show = "showHide1">Click Me!</button>

**ng-hide directive**

Add ng-hide attribute to a HTML button and pass it a model. Bind the model to an checkbox and see the variation.

<input type = "checkbox" ng-model = "showHide2">Hide Button

<button ng-hide = "showHide2">Click Me!</button>

**ng-click directive**

Add ng-click attribute to a HTML button and update a model. Bind the model to html and see the variation.

<p>Total click: {{ clickCounter }}</p>

<button ng-click = "clickCounter = clickCounter + 1">Click Me!</button>

**Example**

Following example will showcase all the above mentioned directives.

*testAngularJS.htm*

|  |
| --- |
| <html> <head> <title>AngularJS HTML DOM</title> </head> <body> <h2>AngularJS Sample Application</h2> <div ng-app = ""> <table border = "0"> <tr> <td><input type = "checkbox" ng-model = "enableDisableButton">Disable Button</td> <td><button ng-disabled = "enableDisableButton">Click Me!</button></td> </tr> <tr> <td><input type = "checkbox" ng-model = "showHide1">Show Button</td> <td><button ng-show = "showHide1">Click Me!</button></td> </tr> <tr> <td><input type = "checkbox" ng-model = "showHide2">Hide Button</td> <td><button ng-hide = "showHide2">Click Me!</button></td> </tr> <tr> <td><p>Total click: {{ clickCounter }}</p></td> <td><button ng-click = "clickCounter = clickCounter + 1">Click Me!</button></td> </tr> </table> </div> <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script> </body></html> |

AngularJS enriches form filling and validation. We can use ng-click to handle AngularJS click on button and use $dirty and $invalid flags to do the validations in seemless way. Use novalidate with a form declaration to disable any browser specific validation. Forms controls makes heavy use of Angular events. Let's have a quick look on events first.

**Events**

AngularJS provides multiple events which can be associated with the HTML controls. For example ng-click is normally associated with button. Following are supported events in Angular JS.

* ng-click
* ng-dbl-click
* ng-mousedown
* ng-mouseup
* ng-mouseenter
* ng-mouseleave
* ng-mousemove
* ng-mouseover
* ng-keydown
* ng-keyup
* ng-keypress
* ng-change

**ng-click**

Reset data of a form using on-click directive of a button.

<input name = "firstname" type = "text" ng-model = "firstName" required>

<input name = "lastname" type = "text" ng-model = "lastName" required>

<input name = "email" type = "email" ng-model = "email" required>

<button ng-click = "reset()">Reset</button>

<script>

 function studentController($scope) {

 $scope.reset = function(){

 $scope.firstName = "enosis";

 $scope.lastName = "learning";

 $scope.email = "courses@enosislearning.com";

 }

 $scope.reset();

 }

</script>

**Validate data**

Following can be used to track error.

* **$dirty** − states that value has been changed.
* **$invalid** − states that value entered is invalid.
* **$error** − states the exact error.

**Example**

Following example will showcase all the above mentioned directives.

*testAngularJS.htm*

<html>

 <head>

 <title>Angular JS Forms</title>

 <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

 <style>

 table, th , td {

 border: 1px solid grey;

 border-collapse: collapse;

 padding: 5px;

 }

 table tr:nth-child(odd) {

 background-color: #f2f2f2;

 }

 table tr:nth-child(even) {

 background-color: #ffffff;

 }

 </style>

 </head>

 <body>

 <h2>AngularJS Sample Application</h2>

 <div ng-app = "mainApp" ng-controller = "studentController">

 <form name = "studentForm" novalidate>

 <table border = "0">

 <tr>

 <td>Enter first name:</td>

 <td><input name = "firstname" type = "text" ng-model = "firstName" required>

 <span style = "color:red" ng-show = "studentForm.firstname.$dirty && studentForm.firstname.$invalid">

 <span ng-show = "studentForm.firstname.$error.required">First Name is required.</span>

 </span>

 </td>

 </tr>

 <tr>

 <td>Enter last name: </td>

 <td>

<input name = "lastname" type = "text" ng-model = "lastName" required>

 <span style = "color:red" ng-show = "studentForm.lastname.$dirty && studentForm.lastname.$invalid">

 <span ng-show = "studentForm.lastname.$error.required">Last Name is required.</span>

 </span>

 </td>

 </tr>

 <tr>

 <td>Email: </td>

<td><input name = "email" type = "email" ng-model = "email" length = "100" required>

 <span style = "color:red" ng-show = "studentForm.email.$dirty && studentForm.email.$invalid">

 <span ng-show = "studentForm.email.$error.required">Email is required.</span>

 <span ng-show = "studentForm.email.$error.email">Invalid email address.</span>

 </span>

</td>

 </tr>

 <tr>

 <td>

 <button ng-click = "reset()">Reset</button>

 </td>

 <td>

 <button ng-disabled = "studentForm.firstname.$dirty &&

 studentForm.firstname.$invalid || studentForm.lastname.$dirty &&

 studentForm.lastname.$invalid || studentForm.email.$dirty &&

 studentForm.email.$invalid" ng-click="submit()">Submit</button>

 </td>

 </tr>

 </table>

 </form>

 </div>

 <script>

 var mainApp = angular.module("mainApp", []);

 mainApp.controller('studentController', function($scope) {

 $scope.reset = function(){

 $scope.firstName = "Enosis";

 $scope.lastName = "Learning";

 $scope.email = "courses@enosislearning.com ";

 }

 $scope.reset();

 });

 </script>

 </body>

</html>

**Required**

Use the HTML5 attribute required to specify that the input field must be filled out:

<form name="myForm">
<input name="myInput" ng-model="myInput" required>
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>

**E-mail**

Use the HTML5 type email to specify that the value must be an e-mail:

<form name="myForm">
<input name="myInput" ng-model="myInput" type="email">
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>

**Form State and Input State**

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

* $untouched The field has not been touched yet
* $touched The field has been touched
* $pristine The field has not been modified yet
* $dirty The field has been modified
* $invalid The field content is not valid
* $valid The field content is valid
* $submitted The form is submitted

They are all properties of the input field, and are either true or false.

They are all properties of the form, and are either true or false.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

**Example**

Show an error message if the field has been touched AND is empty:

<input name="myName" ng-model="myName" required>
<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The name is required.</span>

</style>

**CSS Classes**

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

* ng-untouched The field has not been touched yet
* ng-touched The field has been touched
* ng-pristine The field has not been  modified yet
* ng-dirty The field has been modified
* ng-valid The field content is valid
* ng-invalid The field content is not valid
* ng-valid-*key* One *key* for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
* ng-invalid-*key* Example: ng-invalid-required

The classes are removed if the value they represent is false.

Add styles for these classes to give your application a better and more intuitive user interface.

**Example**

Apply styles, using standard CSS:

<style>

input.ng-invalid {
    background-color: pink;
}
input.ng-valid {
    background-color: lightgreen;
}

**Forms can also be styled:**

**Example :** Apply styles for unmodified (pristine) forms, and for modified forms:

<style>

form.ng-pristine {
    background-color: lightblue;
}
form.ng-dirty {
    background-color: pink;
}

</style>

**Custom Validation**

To create your own validation function is a bit more tricky. You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

**Example:**

Create your own directive, containing a custom validation function, and refer to it by using my-directive.

The field will only be valid if the value contains the character "e":

<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
<script>

var app = angular.module('myApp', []);
app.directive('myDirective', function() {
  return {
    require: 'ngModel',
    link: function(scope, element, attr, mCtrl) {
      function myValidation(value) {
        if (value.indexOf("e") > -1) {
          mCtrl.$setValidity('charE', true);
        } else {
          mCtrl.$setValidity('charE', false);
        }
        return value;
      }
      mCtrl.$parsers.push(myValidation);
    }
  };
});

</script>

**Example Explained:**

In HTML, the new directive will be referred to by using the attribute my-directive.

In the JavaScript we start by adding a new directive named myDirective.

Remember, when naming a directive, you must use a camel case name, myDirective, but when invoking it, you must use - separated name, my-directive.

Then, return an object where you specify that we require  ngModel, which is the ngModelController.

Make a linking function which takes some arguments, where the fourth argument, mCtrl, is the ngModelController,

Then specify a function, in this case named myValidation, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either true or false.

At last, mCtrl.$parsers.push(myValidation); will add the myValidation function to an array of other functions, which will be executed every time the input value changes.

**Validation Example**

<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<h2>Validation Example</h2>
<form  ng-app="myApp"  ng-controller="validateCtrl"
name="myForm" novalidate>
<p>Username:<br>
  <input type="text" name="user" ng-model="user" required>
  <span style="color:red" ng-show="myForm.user.$dirty && myForm.user.$invalid">
  <span ng-show="myForm.user.$error.required">Username is required.</span>
  </span>
</p>
<p>Email:<br>
  <input type="email" name="email" ng-model="email" required>
  <span style="color:red" ng-show="myForm.email.$dirty && myForm.email.$invalid">
  <span ng-show="myForm.email.$error.required">Email is required.</span>
  <span ng-show="myForm.email.$error.email">Invalid email address.</span>
  </span>
</p>
<p>
  <input type="submit"
  ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
  myForm.email.$dirty && myForm.email.$invalid">
</p>
</form>
<script>
var app = angular.module('myApp', []);
app.controller('validateCtrl', function($scope) {
    $scope.user = 'John Doe';
    $scope.email = 'john.doe@gmail.com';
});
</script>
</body>
</html>

Before the release of ngMessages, developers were forced to rely on directives such as ng-class and ng-show to display these errors. This resulted in cluttered, repetitive code.

This will be our basic template:

<!DOCTYPE html>

<html lang="en">

 <head>

 <meta charset="utf-8" />

 <title>ngMessages demo</title>

 </head>

 <body ng-app="app">

 <script src="path/to/angular.min.js"></script>

 <script src="path/to/angular-messages.min.js"></script>

 <script>

 var app = angular.module('app', ['ngMessages']);

 </script>

 </body>

</html>

Next we are going to create a form with the following fields:

1. First Name
2. Last Name
3. Email Address
4. Phone Number
5. Message

I will add a required attribute to all of the fields (as they will be compulsory), as well as using an ng-model directive to bind them to properties on the current scope.

<form name="exampleForm" class="myForm" required>

 <label>First Name:</label>

 <input type="text" name="userFirstName" ng-model="firstName" required />

 <label>Last Name:</label>

 <input type="text" name="userLastName" ng-model="lastName" required />

 <label>Email Address:</label>

 <input type="email" name="userEmail" ng-model="email" required />

 <label>Phone Number:</label>

 <input type="email" name="userPhoneNumber" ng-model="phoneNumber" required />

 <label>User Message:</label>

 <textarea type="text" name="userMessage" ng-model="message" required></textarea>

</form>

**Enter ngMessages**

Now let’s dive into the code and check out how ngMessages allows us to use the attributes we specify on the inputs for simple form validation.

**Required Fields**

The first thing you should take note of is that the form is named exampleForm. When using the ng-messages directive, we pass it an angular expression evaluating to a key/value object (typically the $error object on an ngModel instance). In our case this will be the name of the form, chained to the name attribute of the respective form field, chained to the aforementioned $error object.

<div ng-messages="exampleForm.userFirstName.$error">

Once that is done, you just need to nest a div containing an ng-message attribute inside of the ngMessages div. The value passed to the ng-message attribute will depend upon the directives we added to the input field; in this case, the value will be required.

That will give us:

<label>First Name:</label>

<input type="text" name="userFirstName" ng-model="firstName" required />

<div ng-messages="exampleForm.userFirstName.$error">

 <div ng-message="required">This field is required</div>

</div>

<label>Last Name:</label>

<input type="text" name="userLastName" ng-model="lastName" required />

<div ng-messages="exampleForm.userLastName.$error">

 <div ng-message="required">This field is required</div>

</div>

**Email**

Next, we want to make sure the user enters a valid e-mail address. Luckily, HTML5 makes this a pretty easy task. In the previous example, the name attribute of the input fields took the value of text, however email addresses take the value of email. Just like the example above, you simply chain the form name, input name, and error validator, and pass it to the ng-messages attribute, in order to activate the error messages. We can then add an additional div containing the ng-message attribute that takes the value of email. By doing so, we can target the email value just like we target the required value.

<label>Email Address:</label>

<input type="email" name="userEmail" ng-model="email" required />

<div ng-messages="exampleForm.userEmail.$error">

 <div ng-message="required">This field is required</div>

 <div ng-message="email">Your email address is invalid</div>

</div>

**Regular Expressions**

As we continue along, we will be using ng-pattern directive to validate the user’s phone number. For those who are unfamiliar with ng-pattern, it is used to ensure an input field matches the regular expression that is passed into the attribute. In this case, we will be validating the user’s phone number by passing the following regex into the ng-pattern attribute:

/^[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$/

This will match the following formats (and ensure we accept a valid ten-digit phone number):

(123) 456-7890

123-456-7890

123.456.7890

1234567890

+31636363634

075-63546725

The ng-pattern error message is activated by passing in the value of pattern into ng-message, which can be seen in the code below:

<label>Phone Number:</label>

<input type="email" name="userPhoneNumber" ng-model="phoneNumber"

 ng-pattern="/^[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$/"

 required/>

<div ng-messages="exampleForm.userPhoneNumber.$error">

 <div ng-message="required">This field is required</div>

 <div ng-message="pattern">Must be a valid 10 digit phone number</div>

</div>

**Minlength and Maxlength**

The final thing we want to do is validate the user’s message. The name of this field is userMessage and it is required just like the examples above. This time however, we will utilize the ng-minlength and ng-maxlength attributes. Both of these attributes take integer values which will equal a set number of characters — the ng-minlength attribute is used to set the number of characters a user is limited to, whereas the ng-maxlength attribute sets the maximum numbers of characters that a user is allowed to enter.

In the example below, you will see that we are passing error messages into both the minlength and maxlength values. By doing this, the error message for the minlength value will appear until the user reaches the desired number of characters. Additionally, the error message defined for maxlength will appear once the user surpasses the set number of characters passed into the ng-maxlength attribute.

<label>User Message:</label>

<textarea type="text" name="userMessage" ng-model="message"

 ng-minlength="100" ng-maxlength="1000" required>

</textarea>

<div ng-messages="exampleForm.userMessage.$error">

 <div ng-message="required">This field is required</div>

 <div ng-message="minlength">Message must be over 100 characters</div>

 <div ng-message="maxlength">Message must not exceed 1000 characters</div>

</div>

Now that we have validated all of the necessary fields, you should have a fully functional form that displays whatever message you desire based on the input tags for your form!

**A Final Tweak**

One of the best things about ngMessages, and Angular in general, is the ability it gives you to utilize different directives together. For instance, you can hide the error messages by adding the statement ng-if="exampleForm.inputName.$dirty after the ng-messages directive. To show you how this is done, I have added the ng-if statement to the userMessage section of the form that we have just created. Now, there no error messages until the field is touched, or dirty.

<label>User Message:</label>

<textarea type="text" name="userMessage" ng-model="message"

 ng-minlength="100" ng-maxlength="1000" required>

</textarea>

<div ng-messages="exampleForm.userMessage.$error"

 ng-if="exampleForm.userMessage.$dirty">

 <div ng-message="required">This field is required</div>

 <div ng-message="minlength">Message must be over 100 characters</div>

 <div ng-message="maxlength">Message must not exceed 1000 characters</div>

</div>