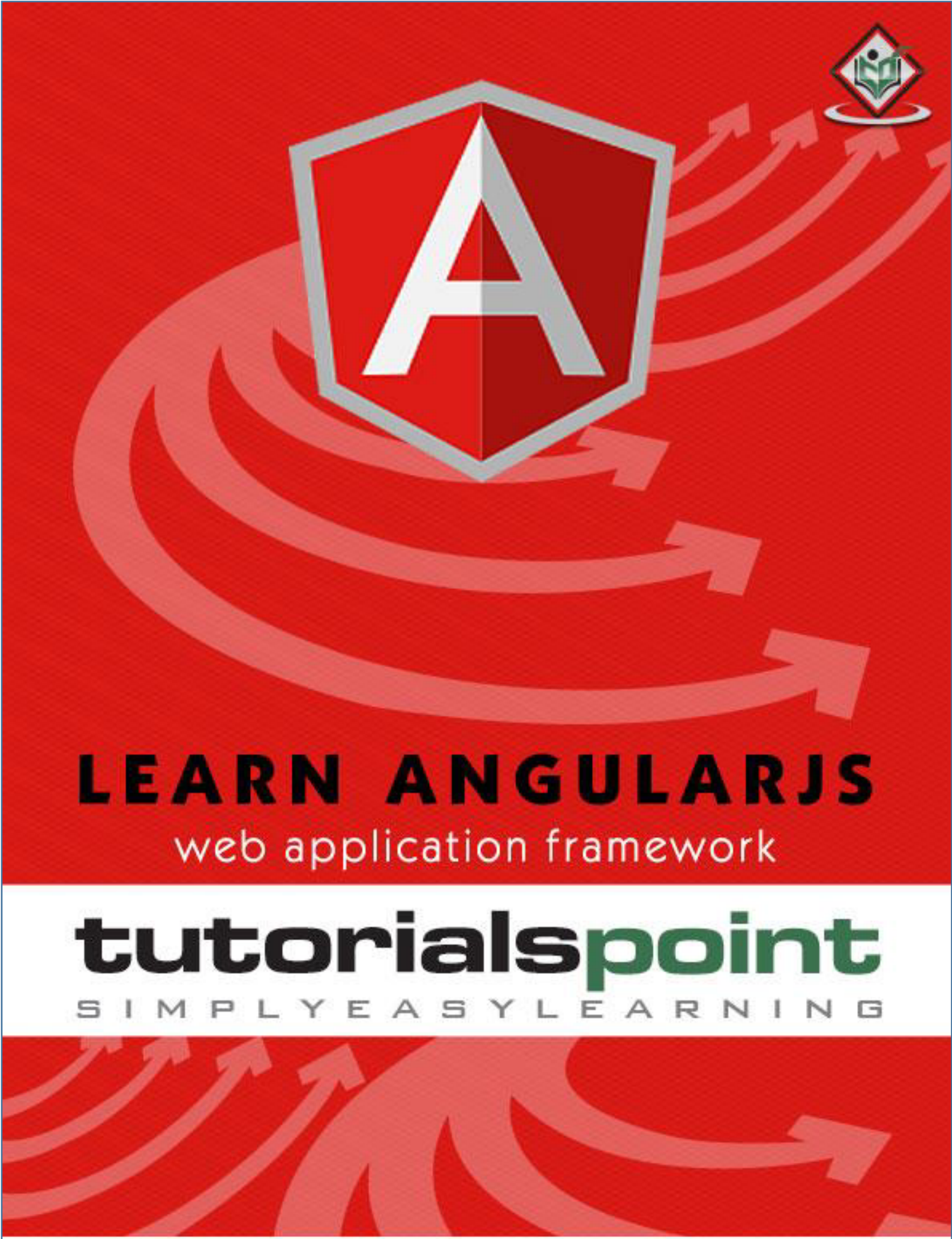


LEARN ANGULARJS

web application framework

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

AngularJS is a very powerful JavaScript library. It is used in Single Page Application (SPA) projects. It extends HTML DOM with additional attributes and makes it more responsive to user actions. 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.

This tutorial teaches you basics of AngularJS and its programming concepts. It describes the components of AngularJS with suitable examples.

Audience

This tutorial is designed for software professionals who are willing to learn AngularJS programming in simple and easy steps. After completing this tutorial, you will be at intermediate level of expertise from where you can take yourself to higher level of expertise.

Prerequisites

You should have basic understanding of scripting language such as JavaScript, and any text editor. You should also know the basic web technologies such as HTML, CSS, AJAX etc. for learning to develop web applications using Angular JS.

Disclaimer & Copyright

Contents

1. OVERVIEW

AngularJS is an open source, JavaScript based web application development framework.

Definition of AngularJS as put by its official documentation is as follows:

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

It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google.

General Features

The most important general features of AngularJS are:

- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
- Applications written in AngularJS are 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, high performance, and easy-to-maintain web applications.

Core Features

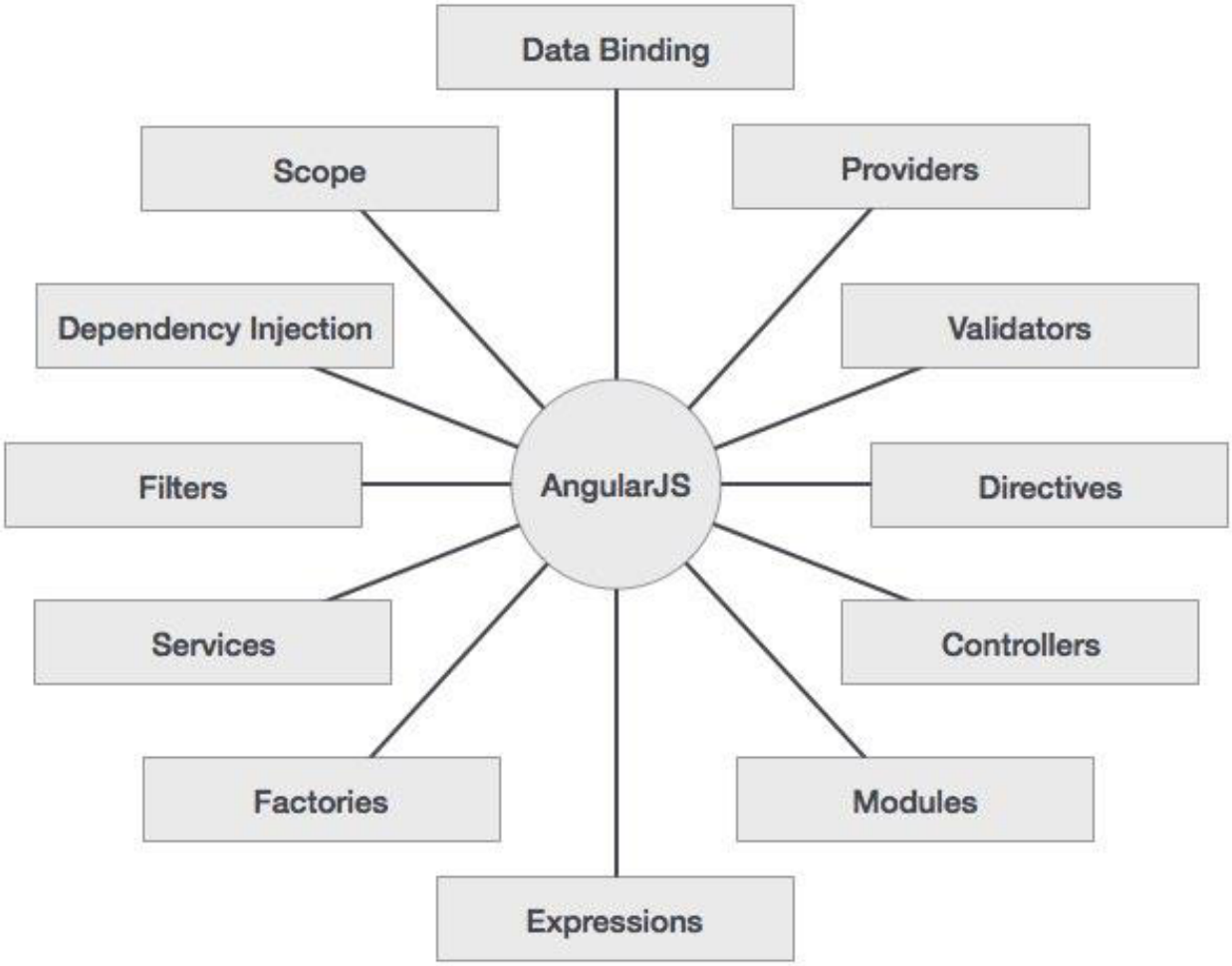
The most important core features of AngularJS are:

- **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 bound to a particular scope.
- **Services:** AngularJS comes with several built-in services such as $http 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 such as ngBind, ngModel etc.
- **Templates:**These are the rendered view with information from the controller and model. These can be a single file (such as index.html) or multiple views in one page using *partials*.
- **Routing:** It is concept of switching views.
- **Model View Whatever:** MVW 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 to create,understand, and test the applications easily.

Concepts

The following diagram depicts some important parts of AngularJS which we will discuss in detail in the subsequent chapters.



Advantages of AngularJS

The advantages of AngularJS are:

- AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
- AngularJS provides data binding capability to HTML. Thus, it gives 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, the developers can achieve more functionality with short code.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

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

Disadvantages of AngulaJS

Though AngularJS comes with a lot of merits, here are some points of concern:

- **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 the user of your application disables JavaScript, then nothing would be visible, except the basic page.

AngularJS Directives

The AngularJS framework can be divided into three major parts:

- **ng-app** : This directive defines and links an AngularJS application to HTML.
- **ng-model** : This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** : This directive binds the AngularJS application data to HTML tags.

2. ENVIRONMENT

This chapter describes how to set up AngularJS library to be used in web application development. It also briefly describes the directory structure and its contents.

When you open the link https://angularjs.org/, you will see there are two options to download AngularJS library:

- **View on GitHub**- By clicking on this button, you are diverted to GitHub and get all the latest scripts.
- **Download**- By clicking on this button, a screen you get to see a dialog box shown as:

This screen offers various options for selecting Angular JS as follows:

- Downloading and hosting files locally
  - There are two different options : Legacy and Latest. The names themselves are self descriptive. The Legacy has version less than 1.2.x and the Latest come with version 1.3.x.
  - We can also go with the minimized, uncompressed, or zipped version.
- CDN access: You also have access to a CDN. The CDN gives you access around the world to regional data centres. In this case, the Google host. This means, using CDN transfers the responsibility of hosting files from your own servers to a series of external ones. This also offers an advantage that if the visitor of your web page has already downloaded a copy of AngularJS from the same CDN, there is no need to re-download it.

We are using the CDN versions of the library throughout this tutorial.

Example

Now let us write a simple example using AngularJS library. Let us create an HTML file*myfirstexample.html* shown as below:

```
<!doctype html>

<html>
```

```
   <head>

      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-
beta.17/angular.min.js"></script>

   </head>

   <body ng-app="myapp">

      <div ng-controller="HelloController" >

         <h2>Welcome {{helloTo.title}} to the world of Tutorialspoint!</h2>

      </div>

      <script>

         angular.module("myapp", [])

         .controller("HelloController", function($scope) {

            $scope.helloTo = {};

            $scope.helloTo.title = "AngularJS";

         });

      </script>

   </body>

</html>
```

Let us go through the above code in detail:

Include AngularJS

We include the AngularJS JavaScript file in the HTML page so that we can use it:

```
<head>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</head>
```

You can check the latest version of AngularJS on its official website.

Point to AngularJS app

Next, it is required to tell which part of HTML contains the AngularJS app. You can do this by adding the *ng-app* attribute to the root HTML element of the AngularJS app. You can either add it to *html* element or *body* element as shown below:

```
<body ng-app="myapp">

</body>
```

View

The view is this part:

```
<div ng-controller="HelloController" >

   <h2>Welcome {{helloTo.title}} to the world of Tutorialspoint!</h2>

</div>
```

ng-controller tells AngularJS which controller to use with this view. *helloTo.title* tells AngularJS to write the *model* value named helloTo.title in HTML at this location.

Controller

The controller part is:

```
<script>

   angular.module("myapp", [])

   .controller("HelloController", function($scope) {

      $scope.helloTo = {};

      $scope.helloTo.title = "AngularJS";

    });

</script>
```

This code registers a controller function named *HelloController* in the angular module named *myapp*. We will study more about modules and controllers in their respective chapters. The controller function is registered in angular via the angular.module(...).controller(...) function call.

The $scope parameter *model* is passed to the controller function. The controller function adds a *helloTo* JavaScript object, and in that object it adds a *title* field.

Execution

Save the above code as *myfirstexample.html* and open it in any browser. You get to see the following output:

Welcome AngularJS to the world of Tutorialspoint!

What happens when the page is loaded in the browser ? Let us see:

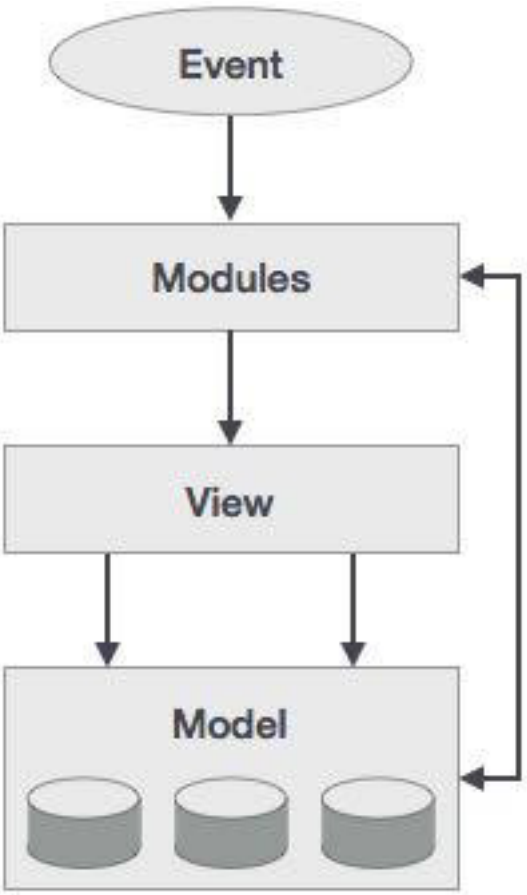
- HTML document is loaded into the browser, and evaluated by the browser.
- AngularJS JavaScript file is loaded, the angular *global* object is created.
- The JavaScript which registers controller functions is executed.
- Next, AngularJS scans through the HTML to search for AngularJS apps as well as views.
- Once the view is located, it connects that view to the corresponding controller function.
- Next, AngularJS executes the controller functions.
- It then renders the views with data from the model populated by the controller. The page is now ready.

3. MVC ARCHITECTURE

Model **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 as 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. In the coming chapters, let us see how AngularJS uses MVC methodology.

4. FIRST APPLICATION

Before creating actual *Hello World !* application using AngularJS, let us see the parts of a AngularJS application. An AngularJS application consists of following three important parts:

- **ng-app** : This directive defines and links an AngularJS application to HTML.
- **ng-model** : This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** : This directive binds the AngularJS Application data to HTML tags.

Creating AngularJS Application

Step 1: Load framework

Being a pure JavaScript framework, it can be added using <Script> tag.

```
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/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 4: Bind the value of above model defined using ng-bind directive.

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

Executing AngularJS Application

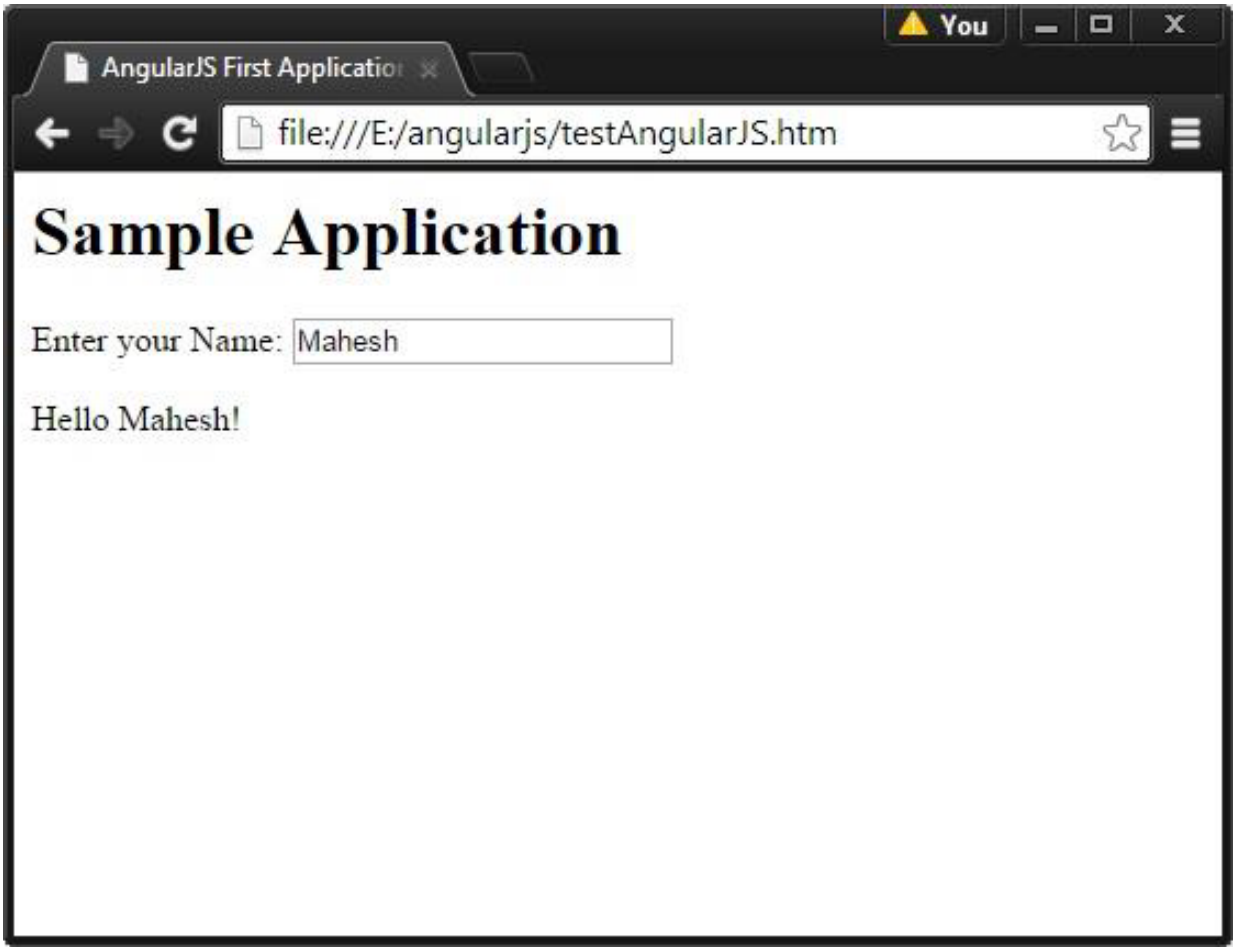
Use the above mentioned three steps in an HTML page.

testAngularJS.htm

```
<html>

<title>AngularJS First Application</title>

<body>

<h1>Sample Application</h1>

<div ng-app="">

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

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

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. Enter your name and see the result.



How AngularJS integrates with HTML

- The ng-app directive indicates the start of AngularJS application.
- The ng-model directive creates a model variable named *name*, which can be used with the HTML page and within the div having ng-app directive.
- The ng-bind then uses the *name* model to be displayed in the HTML tag whenever user enters input in the text box.
- Closing tag indicates the end of AngularJS application.

5. DIRECTIVES

AngularJS directives are used to extend HTML. They are special attributes starting with **ng-**prefix. Let us discuss the following directives:

- **ng-app** - This directive starts an AngularJS Application.
- **ng-init** - This directive initializes application data.
- **ng-model** - This directive defines the model that is variable to be used in AngularJS.
- **ng-repeat** - This directive repeats HTML elements for each item in a collection.

ng-app directive

The ng-app directive starts an AngularJS Application. It defines the root element. It automatically initializes or bootstraps the application when the web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application. In the following example, we define a default AngularJS application using ng-app attribute of a element.

```
<div ng-app="">

...

</div>
```

ng-init directive

The ng-init directive initializes an AngularJS Application data. It is used to assign values to the variables. In the following example, we initialize an array of countries. We use JSON syntax to define the 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

The ng-model directive defines the model/variable to be used in AngularJS Application. In the following example, we define 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 the following example, we iterate over the 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>
```

Example

The following example shows the use of all the above mentioned directives.

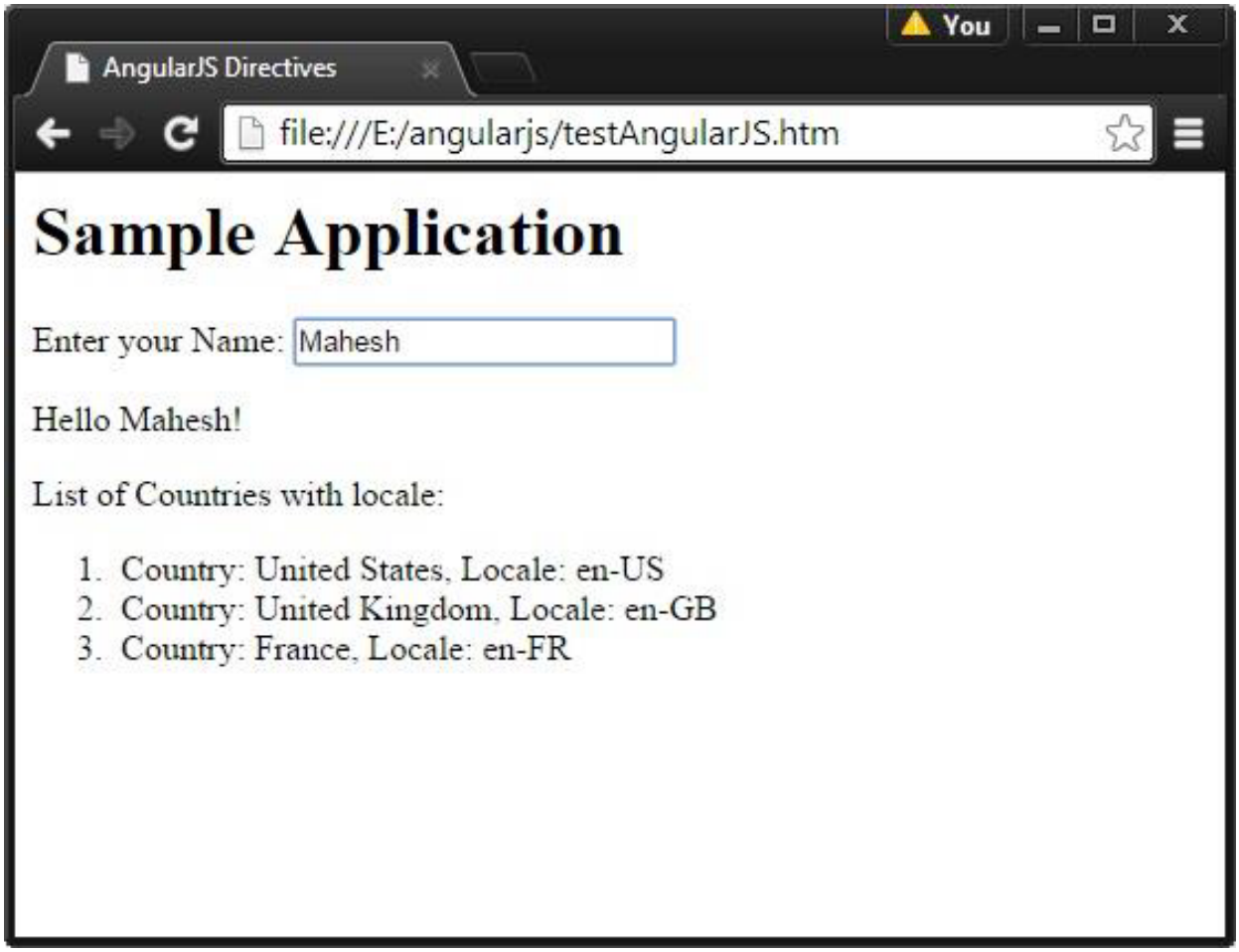
testAngularJS.htm

```
<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/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. Enter your name and see the result.


You
AngularJS Directives
file:///E:/angularjs/testAngularJS.htm
Sample Application
Enter your Name: Mahesh
Hello Mahesh!
List of Countries with locale:
1. Country: United States, Locale: en-US
2. Country: United Kingdom, Locale: en-GB
3. Country: France, Locale: en-FR

6. EXPRESSIONS

Expressions are used to bind application data to HTML. Expressions are written inside double curly braces such as in {{ expression}}. Expressions behave similar to ng-bind directives. AngularJS expressions are pure JavaScript expressions and output the data where they are used.

Using numbers

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

Using String

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

Using Object

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

Using Array

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

Example

The following example shows use of all the above mentioned expressions:

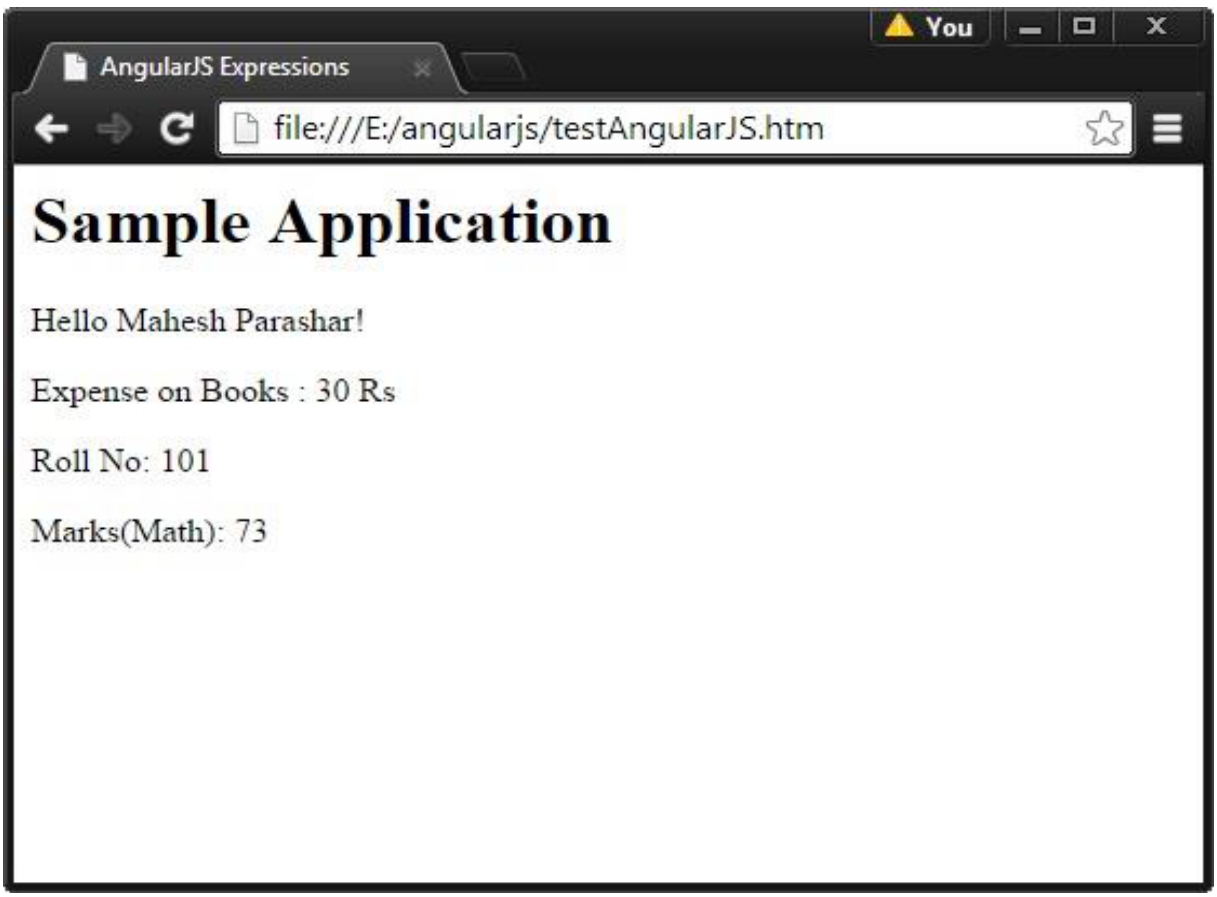
testAngularJS.htm

```
<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/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



7. CONTROLLERS

AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using *ng-controller* directive. A controller is a JavaScript object that contains attributes/properties, and functions. Each controller accepts $scope as a parameter, which refers to the application/module that the controller needs to handle.

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

...

</div>
```

Here, we declare a controller named *studentController*, using ng-controller directive. As a next step, we define it as follows:

```
<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>
```

- The studentController is defined as a JavaScript object with $scope as an argument.
- The $scope refers to application which uses the studentController object.

- The $scope.student is a property of studentController object.
- The firstName and the lastName are two properties of $scope.student object. We pass the default values to them.
- The property fullName is the function of $scope.student object, which returns the combined name.
- In the fullName function, we get the student object and then return the combined name.
- As a note, we can also define the controller object in a separate JS file and refer that file in the HTML page.

Now we can use studentController's student property using ng-model or using expressions as follows:

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

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

<br>

You are entering: {{student.fullName()}}
```

- We bound student.firstName and student.lastname to two input boxes.
- We bound student.fullName() to HTML.
- Now whenever you type anything in first name and last name input boxes, you can see the full name getting updated automatically.

Example

The following example shows the use of controller:

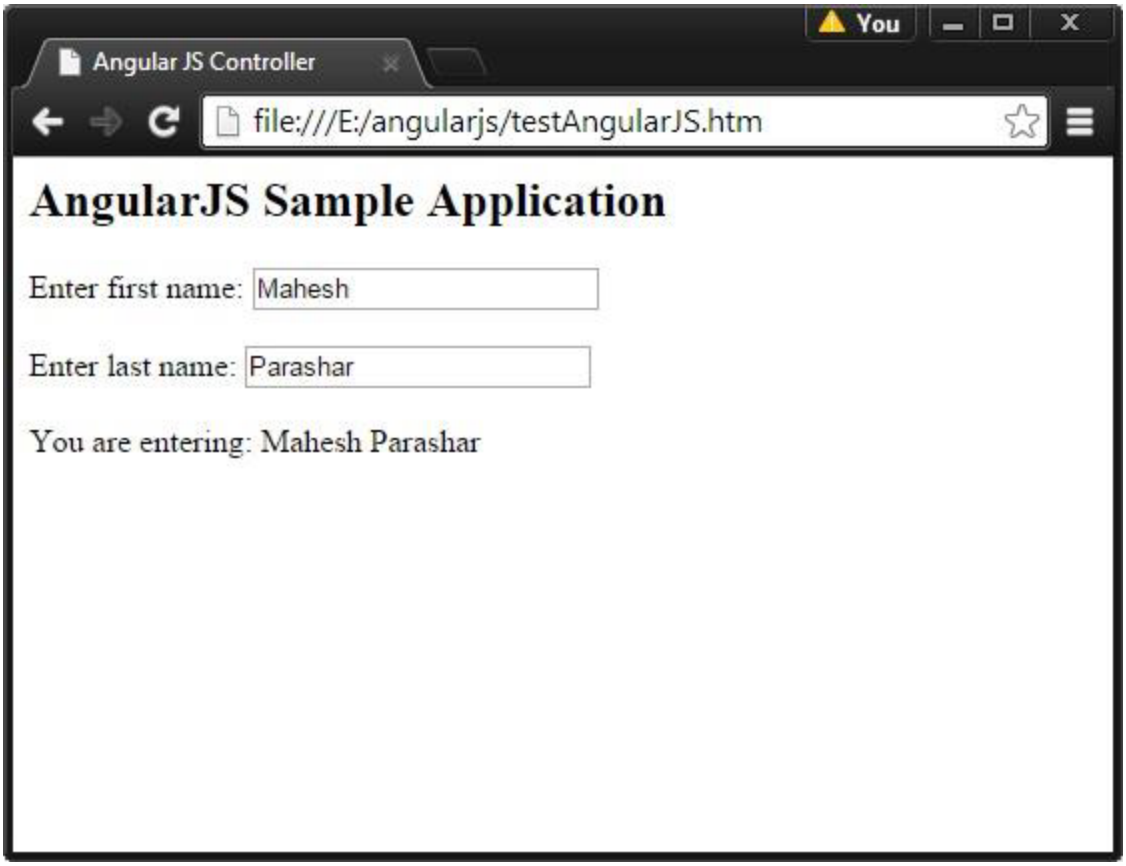
testAngularJS.htm

```
<html>

<head>

<title>Angular JS Controller</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>
```

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

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

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

<br>

You are entering: {{student.fullName()}}

</div>

<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.


You
Angular JS Controller
file:///E:/angularjs/testAngularJS.htm
AngularJS Sample Application
Enter first name: Mahesh
Enter last name: Parashar
You are entering: Mahesh Parashar

8. FILTERS

Filters are used to modify the data. They can be clubbed in expression or directives using pipe (|) character. The following list shows commonly used filters.

S.No.	Name	Description
1	uppercase	converts a text to upper case text.
2	lowercase	converts a text to lower case text.
3	currency	formats text in a currency format.
4	filter	filter the array to a subset of it based on provided criteria.
5	orderby	orders the array based on provided criteria.

Uppercase Filter

This adds uppercase filter to an expression using pipe character. Here, we add uppercase filter to print student name in 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

This adds lowercase filter to an expression using pipe character. Here, we add lowercase filter to print student name in small 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}}
```

Currency Filter

This adds currency filter to an expression that returns a number. Here, we add currency filter to print fees using currency format.

```
Enter fees: <input type="text" ng-model="student.fees">

fees: {{student.fees | currency}}
```

Filter Filter

To display only required subjects, we use subjectName as filter.

```
Enter subject: <input type="text" ng-model="subjectName">

Subject:

<ul>

  <li ng-repeat="subject in student.subjects | filter: subjectName">

    {{ subject.name + ', marks:' + subject.marks }}

  </li>

</ul>
```

Orderby Filter

To order subjects by marks, we use orderBy marks.

```
Subject:

<ul>

  <li ng-repeat="subject in student.subjects | orderBy:'marks'">

    {{ subject.name + ', marks:' + subject.marks }}

  </li>

</ul>
```

Example

The following example shows use of all the above mentioned filters.

testAngularJS.htm

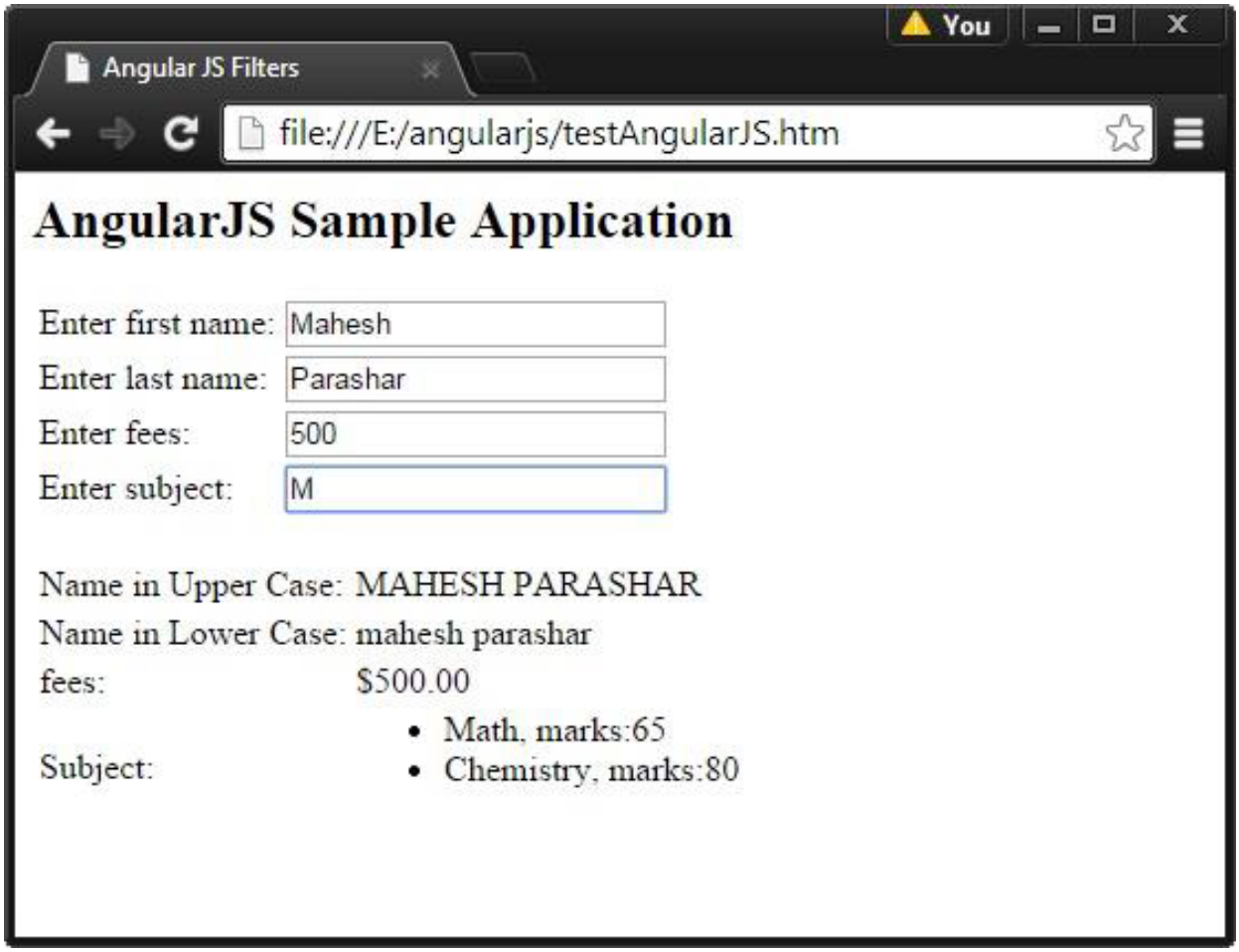
```
<html>

<head>
```

```
<title>Angular JS Filters</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>

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

<table border="0">

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

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

<tr><td>Enter fees: </td><td><input type="text" ng-
model="student.fees"></td></tr>

<tr><td>Enter subject: </td><td><input type="text" ng-
model="subjectName"></td></tr>

</table>

<br/>

<table border="0">

<tr><td>Name in Upper Case: </td><td>{{student.fullName() |
uppercase}}</td></tr>

<tr><td>Name in Lower Case: </td><td>{{student.fullName() |
lowercase}}</td></tr>

<tr><td>fees: </td><td>{{student.fees | currency}}</td></tr>

<tr><td>Subject:</td><td>

<ul>

   <li ng-repeat="subject in student.subjects | filter: subjectName
|orderBy:'marks'">

      {{ subject.name + ', marks:' + subject.marks }}

   </li>

</ul>

</td></tr>

</table>
```

```
</div>

<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},

         {name:'Math',marks:65}

      ],

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. See the result.


You
Angular JS Filters
file:///E:/angularjs/testAngularJS.htm
AngularJS Sample Application
Enter first name: Mahesh
Enter last name: Parashar
Enter fees: 500
Enter subject: M
Name in Upper Case: MAHESH PARASHAR
Name in Lower Case: mahesh parashar
fees: $500.00
Subject:
Math, marks:65
Chemistry, marks:80

9. TABLES

Table data is generally repeatable. The ng-repeat directive can be used to draw table easily. The following example shows the use of ng-repeat directive to draw a table:

```
<table>
   <tr>
      <th>Name</th>
      <th>Marks</th>
   </tr>
   <tr ng-repeat="subject in student.subjects">
      <td>{{ subject.name }}</td>
      <td>{{ subject.marks }}</td>
   </tr>
</table>
```

Table can be styled using CSS Styling.

```
<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>
```

Example

The following example shows use of all the above mentioned directives.

testAngularJS.htm

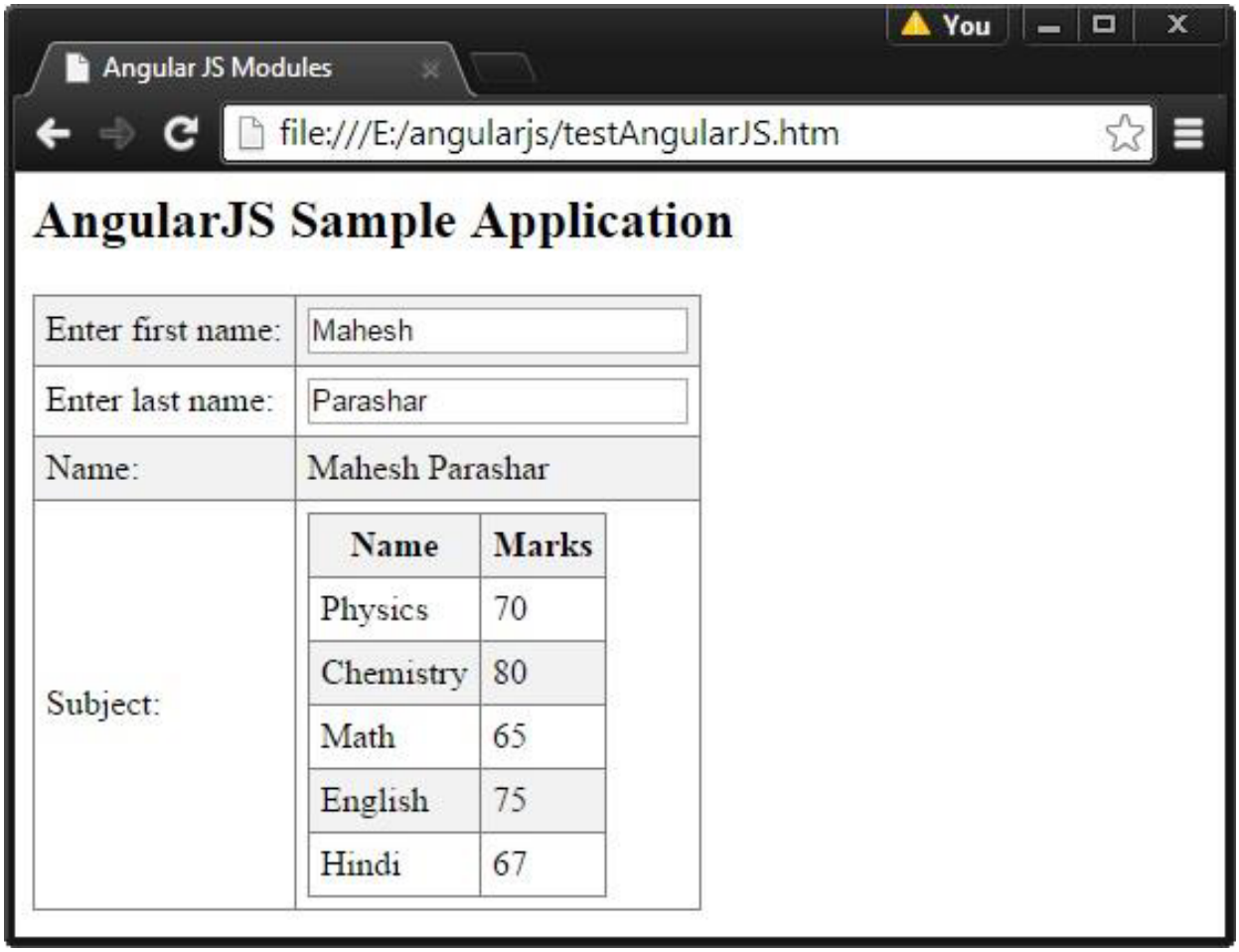
```
<html>
<head>
<title>Angular JS Table</title>
<style>
table, th , td {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
table tr:nth-child(odd) {
  background-color: #f2f2f2;
}
table tr:nth-child(even) {
  background-color: #ffffff;
}
</style>
</head>
<body>
<h2>AngularJS Sample Application</h2>
<div ng-app="" ng-controller="studentController">
<table border="0">
```

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

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

<tr><td>Name: </td><td>{{student.fullName()}}</td></tr>

<tr><td>Subject:</td><td>

<table>

   <tr>

      <th>Name</th>

      <th>Marks</th>

   </tr>

   <tr ng-repeat="subject in student.subjects">

      <td>{{ subject.name }}</td>

      <td>{{ subject.marks }}</td>

   </tr>

</table>

</td></tr>

</table>

</div>

<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},
```

```
         {name:'Math',marks:65},
               {name:'English',marks:75},
               {name:'Hindi',marks:67}
      ],
      fullName: function() {
         var studentObject;
         studentObject = $scope.student;
         return studentObject.firstName + " " + studentObject.lastName;
      }
   };
}
</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.


You
Angular JS Modules
file:///E:/angularjs/testAngularJS.htm
AngularJS Sample Application
Enter first name:
Mahesh
Enter last name:
Parashar
Name:
Mahesh Parashar
Subject:
Name
Marks
Physics
70
Chemistry
80
Math
65
English
75
Hindi
67

10. HTML DOM

The following directives are used to bind application data to attributes of HTML DOM elements:

S.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 an HTML button and pass it a model. Bind the model to a 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 an HTML button and pass it a model. Bind the model to a 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 an HTML button and pass it a model. Bind the model to a 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 an HTML button and update a model. Bind the model to HTML and see the variation.

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

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

Example

The following example shows use of all the above mentioned directives.

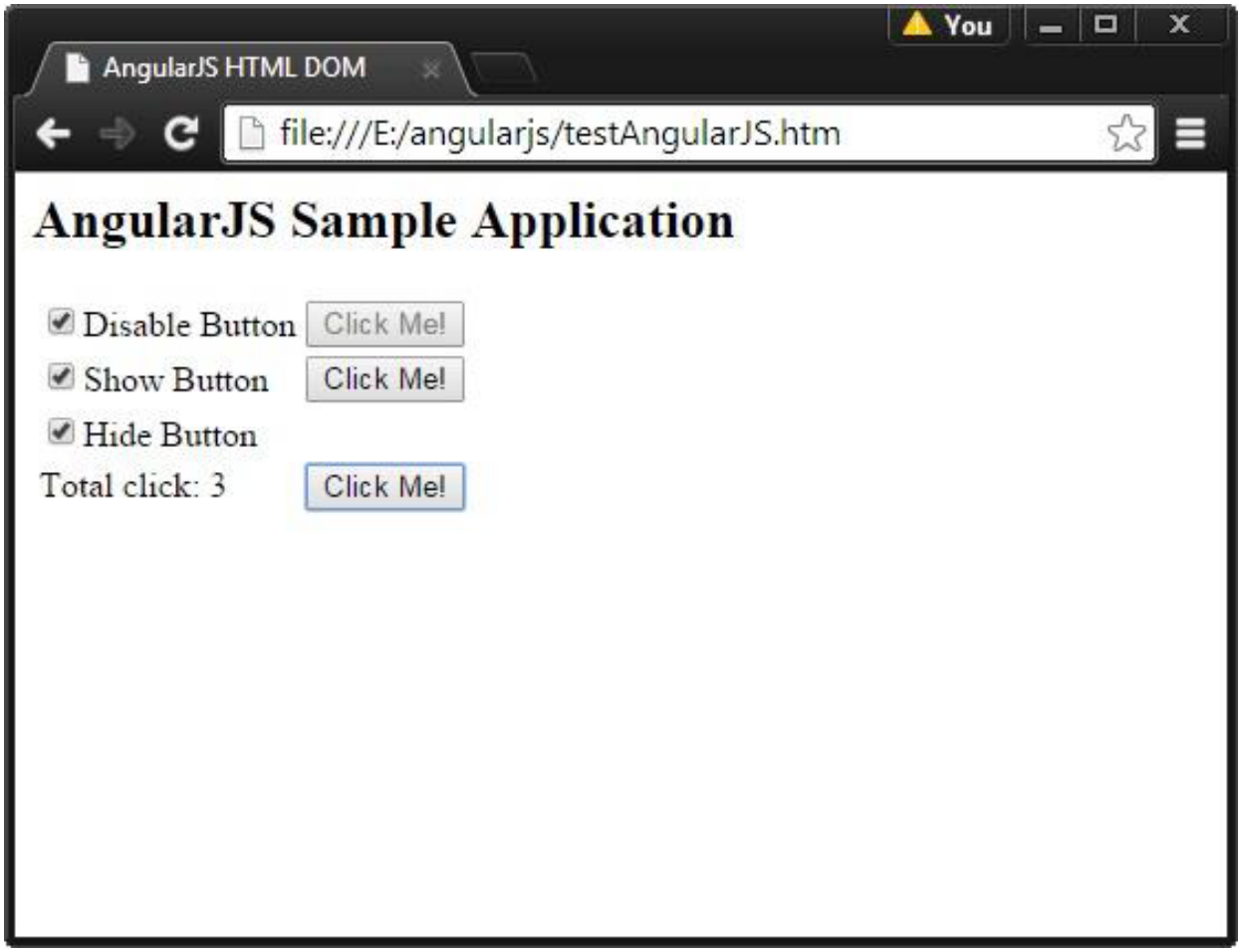
testAngularJS.htm

```
<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="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.


You
AngularJS HTML DOM
file:///E:/angularjs/testAngularJS.htm
AngularJS Sample Application
Disable Button
Click Me!
Show Button
Click Me!
Hide Button
Total click: 3
Click Me!

11. MODULES

AngularJS supports modular approach. Modules are used to separate logic such as services, controllers, application etc. from the code and maintain the code clean. We define modules in separate js files and name them as per the module.js file. In the following example, we are going to create two modules:

- **Application Module** - used to initialize an application with controller(s).
- **Controller Module** - used to define the controller.

Application Module

Here is a file named *mainApp.js* that contains the following code:

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

Here, we declare an application **mainApp** module using angular.module function and pass an empty array to it. This array generally contains dependent modules.

Controller Module

studentController.js

```
mainApp.controller("studentController", function($scope) {
   $scope.student = {
      firstName: "Mahesh",
      lastName: "Parashar",
      fees:500,
      subjects:[
         {name:'Physics',marks:70},
         {name:'Chemistry',marks:80},
         {name:'Math',marks:65},
         {name:'English',marks:75},
         {name:'Hindi',marks:67}
```

```
      ],
      fullName: function() {
         var studentObject;
         studentObject = $scope.student;
         return studentObject.firstName + " " + studentObject.lastName;
      }
   };
});
```

Here, we declare a controller **studentController** module using mainApp.controller function.

Use Modules

```
<div ng-app="mainApp" ng-controller="studentController">
..
<script src="mainApp.js"></script>
<script src="studentController.js"></script>
```

Here, we use application module using ng-app directive, and controller using ng-controller directive. We import the mainApp.js and studentController.js in the main HTML page.

Example

The following example shows use of all the above mentioned modules.

testAngularJS.htm

```
<html>
<head>
<title>Angular JS Modules</title>
<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">

<table border="0">

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

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

<tr><td>Name: </td><td>{{student.fullName()}}</td></tr>

<tr><td>Subject:</td><td>

<table>

   <tr>

      <th>Name</th>

      <th>Marks</th>

   </tr>

   <tr ng-repeat="subject in student.subjects">

      <td>{{ subject.name }}</td>
```

```
      <td>{{ subject.marks }}</td>

   </tr>

</table>

</td></tr>

</table>

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

<script src="mainApp.js"></script>

<script src="studentController.js"></script>

</body>

</html>
```

mainApp.js

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

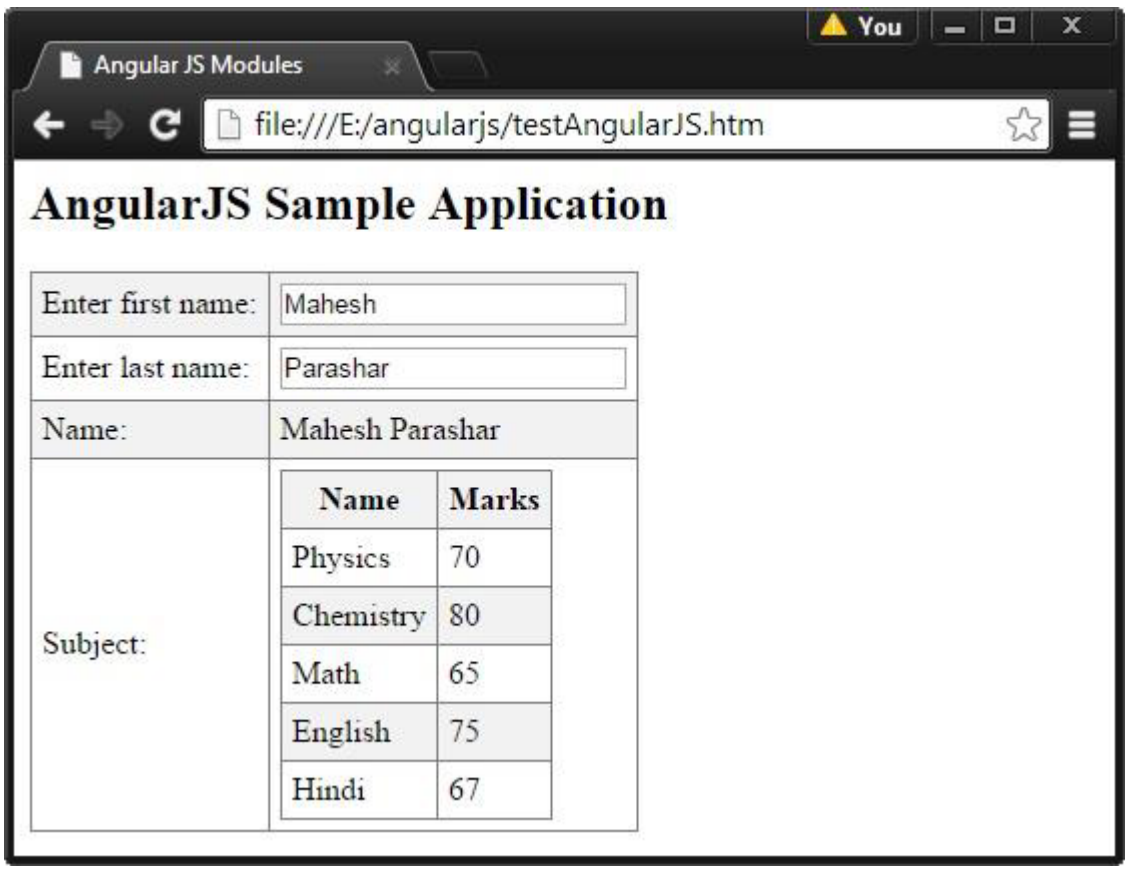
studentController.js

```
mainApp.controller("studentController", function($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},

         {name:'Math',marks:65},

         {name:'English',marks:75},

         {name:'Hindi',marks:67}
```

```
      ],
      fullName: function() {
         var studentObject;
         studentObject = $scope.student;
         return studentObject.firstName + " " + studentObject.lastName;
      }
   };
});
```

Output

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



12. FORMS

AngularJS enriches form filling and validation. We can use ng-click event to handle the click button and use $dirty and $invalid flags to do the validation in a seamless way. Use novalidate with a form declaration to disable any browser-specific validation. The form controls make heavy use of AngularJS events. Let us have a look at the events first.

Events

AngularJS provides multiple events associated with the HTML controls. For example, ng-click directive is generally associated with a button. AngularJS supports the following events:

- 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

Let us go through ng-click:

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 = "Mahesh";

         $scope.lastName = "Parashar";

         $scope.email = "MaheshParashar@tutorialspoint.com";

  }

   $scope.reset();

}

</script>
```

Validate Data

The 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>

<style>

table, th , td {

  border: 1px solid grey;

  border-collapse: collapse;
```

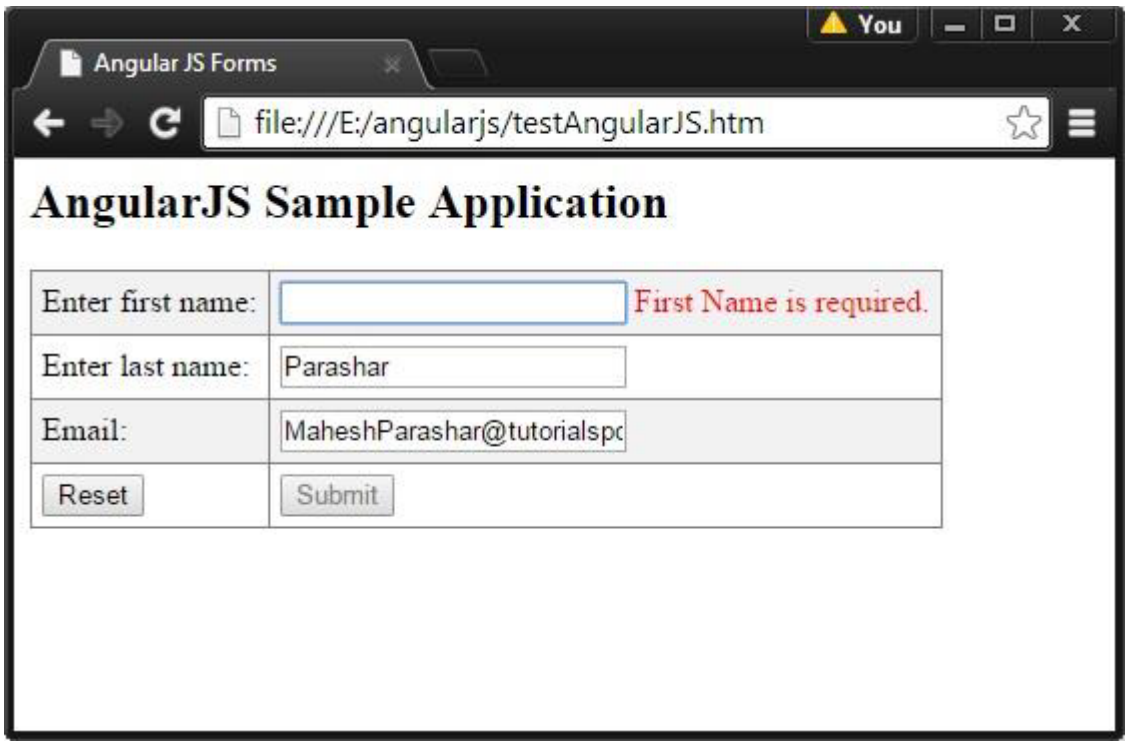
```
  padding: 5px;

}

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="" 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>

function studentController($scope) {

   $scope.reset = function(){

              $scope.firstName = "Mahesh";

              $scope.lastName = "Parashar";

              $scope.email = "MaheshParashar@tutorialspoint.com";

   }

   $scope.reset();

}
```

```
</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



13. INCLUDES

HTML does not support embedding HTML pages within the HTML page. To achieve this functionality, we can use one of the following options:

- **Using Ajax** - Make a server call to get the corresponding HTML page and set it in innerHTML of HTML control.
- **Using Server Side Includes** - JSP, PHP and other web side server technologies can include HTML pages within a dynamic page.

Using AngularJS, we can embed HTML pages within an HTML page using ng-include directive.

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

   <div ng-include="'main.htm'"></div>

   <div ng-include="'subjects.htm'"></div>

</div>
```

Example

tryAngularJS.htm

```
<html>

<head>

<title>Angular JS Includes</title>

<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="" ng-controller="studentController">
<div ng-include="'main.htm'"></div>
<div ng-include="'subjects.htm'"></div>
</div>
<script>
function studentController($scope) {
   $scope.student = {
      firstName: "Mahesh",
      lastName: "Parashar",
      fees:500,
      subjects:[
         {name:'Physics',marks:70},
         {name:'Chemistry',marks:80},
         {name:'Math',marks:65},
         {name:'English',marks:75},
         {name:'Hindi',marks:67}
      ],
      fullName: function() {
```

```
         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

main.htm

```
<table border="0">

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

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

   <tr><td>Name: </td><td>{{student.fullName()}}</td></tr>

</table>
```

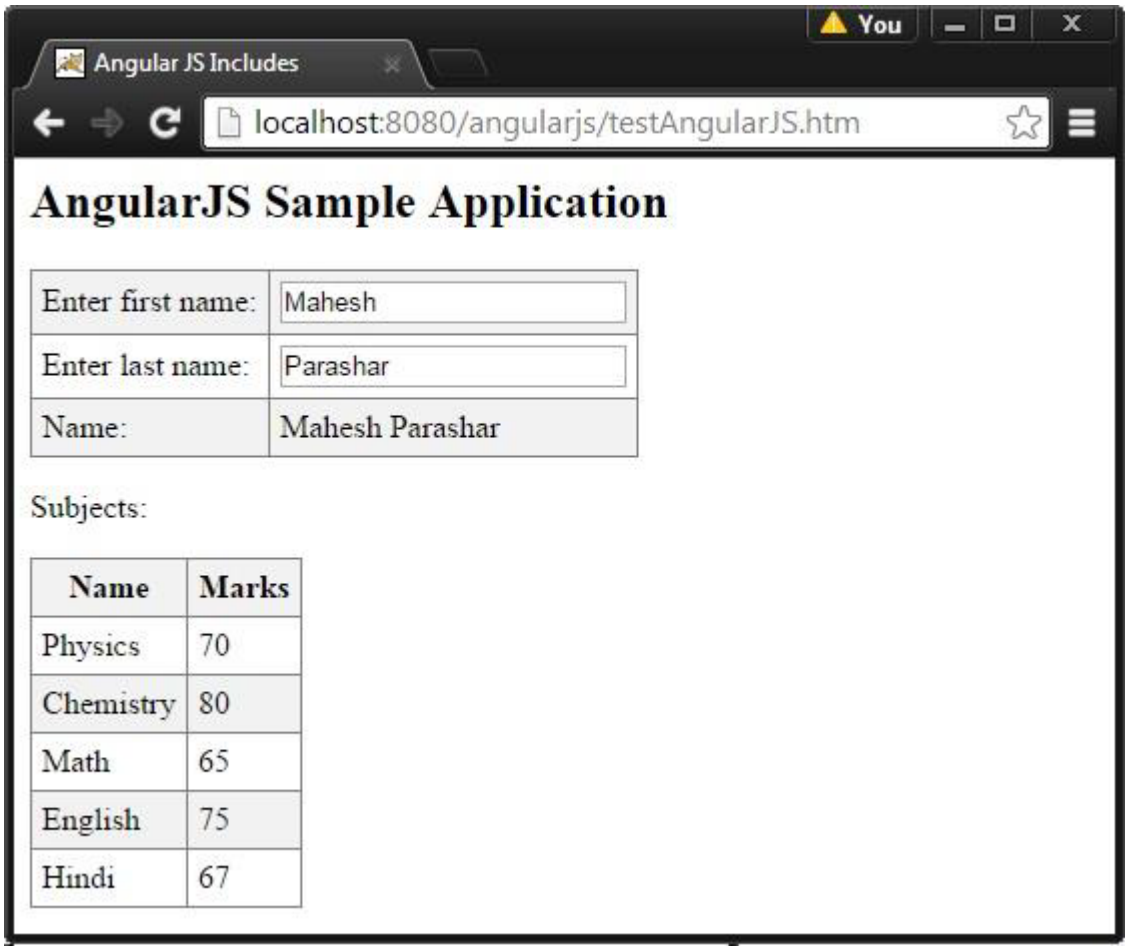
subjects.htm

```
<p>Subjects:</p>

<table>

   <tr>

      <th>Name</th>

      <th>Marks</th>

   </tr>

   <tr ng-repeat="subject in student.subjects">
```

```
      <td>{{ subject.name }}</td>
      <td>{{ subject.marks }}</td>
   </tr>
</table>
```

Output

To execute this example, you need to deploy *testAngularJS.htm*, *main.htm*, and*subjects.htm* to a web server. Open the file *testAngularJS.htm* using the URL of your server in a web browser and see the result.



14. AJAX

AngularJS provides $http control which works as a service to read data from the server. Server makes a database call to get the desired records. AngularJS needs data in JSON format. Once the data is ready, $http can be used to get the data from server in the following manner:

```
function studentController($scope,$http) {

var url="data.txt";

   $http.get(url).success( function(response) {

                              $scope.students = response;

                          });

}
```

Here, the file data.txt contains student records. $http service makes an ajax call and sets response to its property students. *students* model can be used to draw tables in HTML.

Examples

data.txt

```
[

{

"Name" : "Mahesh Parashar",

"RollNo" : 101,

"Percentage" : "80%"

},

{

"Name" : "Dinkar Kad",

"RollNo" : 201,

"Percentage" : "70%"
```

```
},
{
"Name" : "Robert",
"RollNo" : 191,
"Percentage" : "75%"
},
{
"Name" : "Julian Joe",
"RollNo" : 111,
"Percentage" : "77%"
}
]
```

testAngularJS.htm

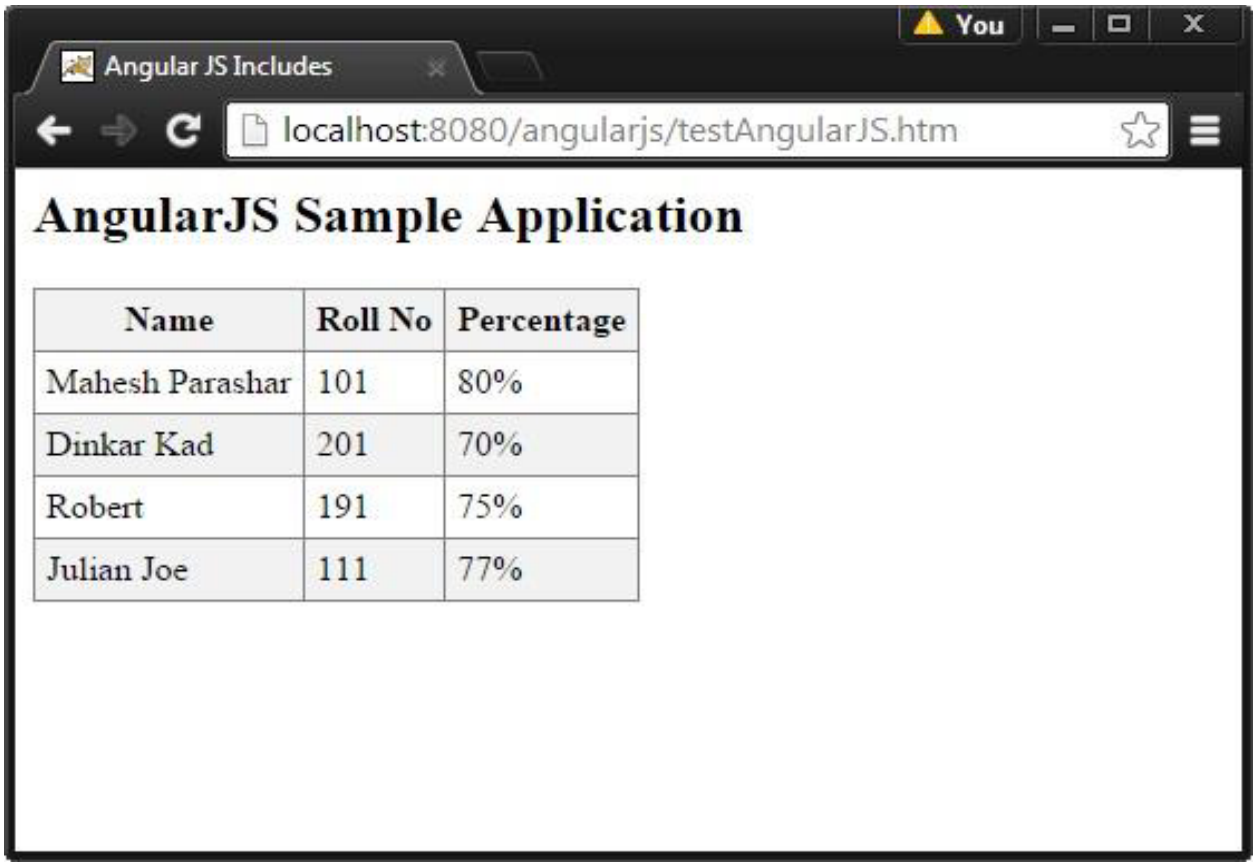
```
<html>
<head>
<title>Angular JS Includes</title>
<style>
table, th , td {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
table tr:nth-child(odd) {
  background-color: #f2f2f2;
}
table tr:nth-child(even) {
  background-color: #ffffff;
```

```
}
</style>
</head>
<body>
<h2>AngularJS Sample Application</h2>
<div ng-app="" ng-controller="studentController">
<table>
   <tr>
      <th>Name</th>
      <th>Roll No</th>
        <th>Percentage</th>
   </tr>
   <tr ng-repeat="student in students">
      <td>{{ student.Name }}</td>
      <td>{{ student.RollNo }}</td>
        <td>{{ student.Percentage }}</td>
   </tr>
</table>
</div>
<script>
function studentController($scope,$http) {
var url="data.txt";
   $http.get(url).success( function(response) {
                           $scope.students = response;
                        });
}
</script>
```

```
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

To execute this example, you need to deploy *testAngularJS.htm* and *data.txt* file to a web server. Open the file *testAngularJS.htm* using the URL of your server in a web browser and see the result.



15. VIEWS

AngularJS supports Single Page Application via multiple views on a single page. To do this, AngularJS has provided ng-view and ng-template directives, and $routeProvider services.

ng-view Directive

The ng-view directive simply creates a place holder where a corresponding view (HTML or ng-template view) can be placed based on the configuration.

Usage

Define a div with ng-view within the main module.

```
<div ng-app="mainApp">

...

   <div ng-view></div>



</div>
```

ng-template Directive

The ng-template directive is used to create an HTML view using *script* tag. It contains *id*attribute which is used by $routeProvider to map a view with a controller.

Usage

Define a script block with type as ng-template within the main module.

```
<div ng-app="mainApp">

...

   <script type="text/ng-template" id="addStudent.htm">

      <h2> Add Student </h2>

         {{message}}
```

```
   </script>

</div>
```

$routeProvider Service

The $routeProvider is a key service which sets the configuration of URLs, maps them with the corresponding HTML page or ng-template, and attaches a controller with the same.

Usage 1

Define a script block with type as ng-template within the main module.

```
<div ng-app="mainApp">

...

   <script type="text/ng-template" id="addStudent.htm">

      <h2> Add Student </h2>

         {{message}}

   </script>


</div>
```

Usage 2

Define a script block with main module and set the routing configuration.

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


      mainApp.config(['$routeProvider',

         function($routeProvider) {

            $routeProvider.

               when('/addStudent', {

                  templateUrl: 'addStudent.htm',
```

```
            controller: 'AddStudentController'
         }).
         when('/viewStudents', {
            templateUrl: 'viewStudents.htm',
            controller: 'ViewStudentsController'
         }).
         otherwise({
            redirectTo: '/addStudent'
         });
   }]);
```

The following points are important to be considered in above example:

- The $routeProvider is defined as a function under config of mainApp module using key as '$routeProvider'.
- The $routeProvider.when defines a URL "/addStudent", which is mapped to "addStudent.htm". addStudent.htm should be present in the same path as main HTML page.If the HTML page is not defined, then ng-template needs to be used with id="addStudent.htm". We used ng-template.
- The "otherwise" is used to set the default view.
- The "controller" is used to set the corresponding controller for the view.

Example

The following example shows use of all the above mentioned directives.

testAngularJS.htm

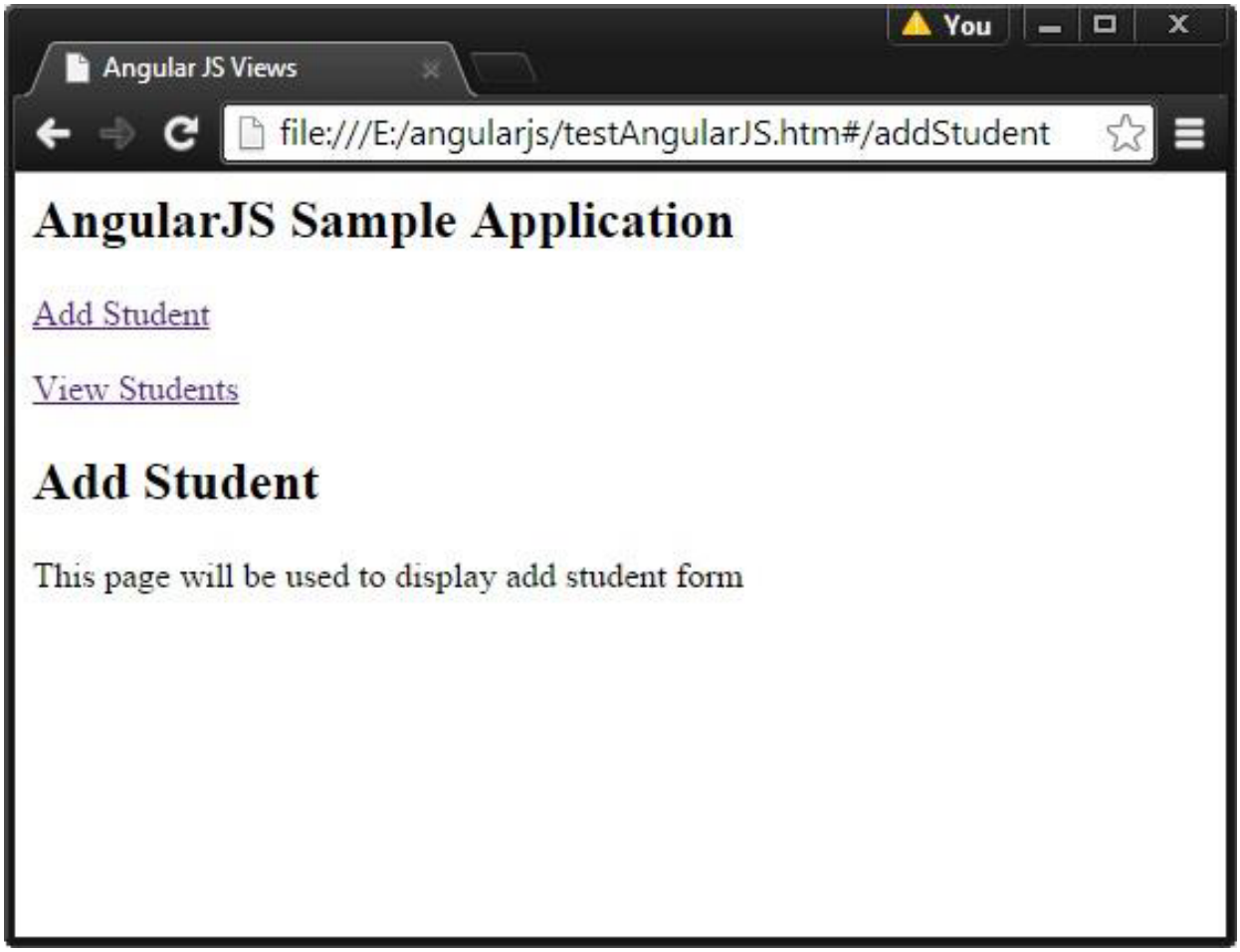
```
<html>
<head>
   <title>Angular JS Views</title>
```

```
   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular-
route.min.js"></script>

</head>

<body>

   <h2>AngularJS Sample Application</h2>

   <div ng-app="mainApp">

      <p><a href="#addStudent">Add Student</a></p>

      <p><a href="#viewStudents">View Students</a></p>

      <div ng-view></div>

      <script type="text/ng-template" id="addStudent.htm">

         <h2> Add Student </h2>

         {{message}}

      </script>

      <script type="text/ng-template" id="viewStudents.htm">

         <h2> View Students </h2>

         {{message}}

      </script>

   </div>



   <script>

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



      mainApp.config(['$routeProvider',

         function($routeProvider) {

            $routeProvider.
```

```
            when('/addStudent', {
               templateUrl: 'addStudent.htm',
               controller: 'AddStudentController'
            }).
            when('/viewStudents', {
               templateUrl: 'viewStudents.htm',
               controller: 'ViewStudentsController'
            }).
            otherwise({
               redirectTo: '/addStudent'
            });
      }]);

      mainApp.controller('AddStudentController', function($scope) {
         $scope.message = "This page will be used to display add student
form";
      });

      mainApp.controller('ViewStudentsController', function($scope) {
         $scope.message = "This page will be used to display all the
students";
      });
   </script>
</body>
</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.


You
Angular JS Views
file:///E:/angularjs/testAngularJS.htm#/addStudent
AngularJS Sample Application
Add Student
View Students
Add Student
This page will be used to display add student form

16. SCOPES

Scope is a special JavaScript object that connects controller with views. Scope contains 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";

      });

</script>
```

The following important points are considered in above example:

- The $scope is passed as first argument to controller during its constructor definition.
- The *$scope.message* and *$scope.type* are the models which are used in the HTML page.
- We assign values to models that are reflected in the application module, whose controller is shapeController.
- We can define functions in $scope.

Scope Inheritance

Scope is controller-specific. If we define nested controllers, then the child controller inherits 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>
```

The following important points are considered in above example:

- We assign values to the models in shapeController.
- We override message in child controller named *circleController*. When *message* is used within the module of controller named *circleController*, the overridden message is used.

Example

The following example shows use of all the above mentioned directives.

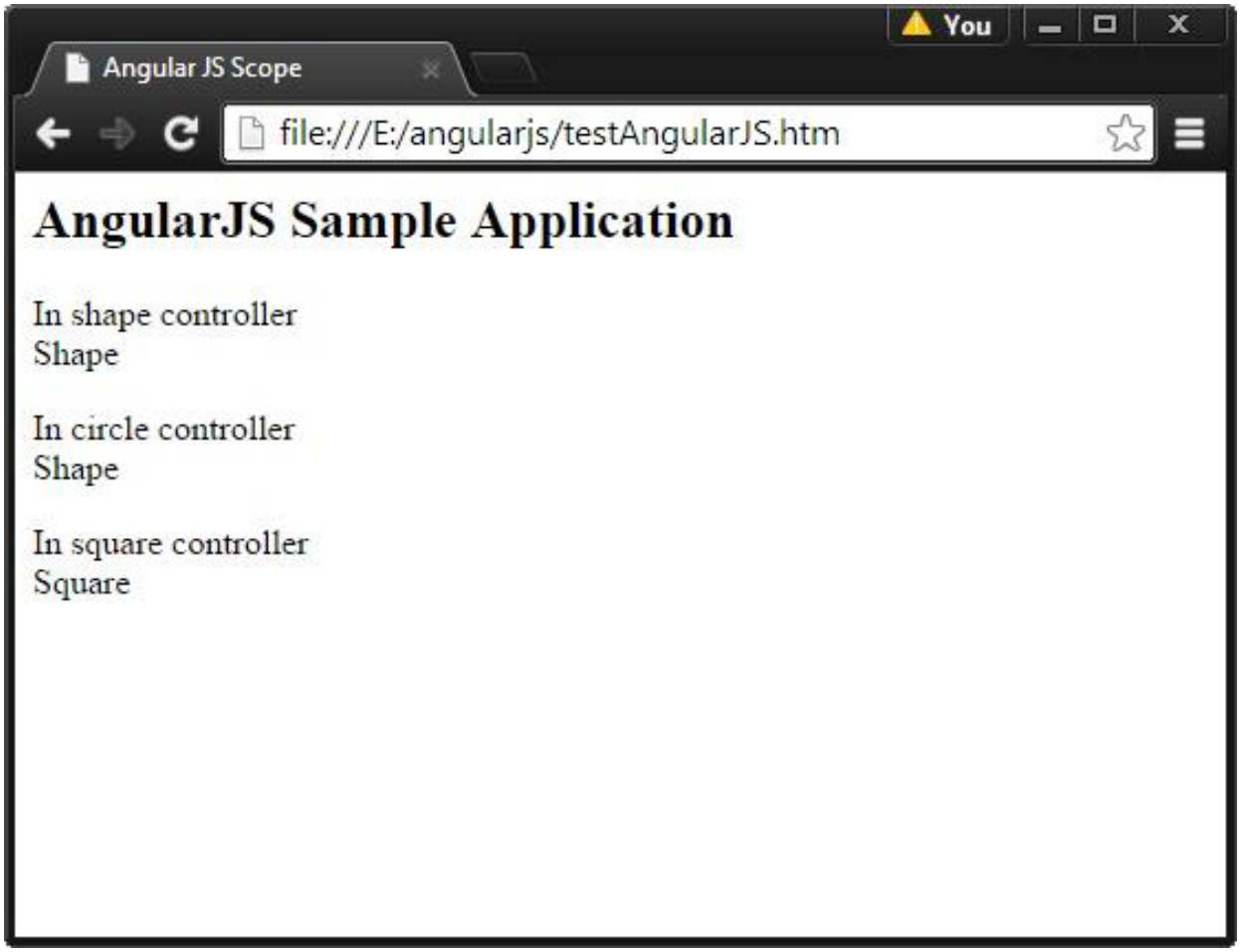
testAngularJS.htm

```
<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/1.2.15/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>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



17. SERVICES

AngularJS supports the concept of *Separation of Concerns* using services architecture. Services are JavaScript functions, which are responsible to perform only specific tasks. This makes them individual entities which are maintainable and testable. The controllers and filters can call them on requirement basis. Services are normally injected using dependency injection mechanism of AngularJS.

AngularJS provides many inbuilt services. For example, $http, $route, $window, $location etc. Each service is responsible for a specific task such as the $http is used to make ajax call to get the server data, the $route is used to define the routing information, and so on. The inbuilt services are always prefixed with $ symbol.

There are two ways to create a service:

- Factory
- Service

Using Factory Method

In this method, we first define a factory and then assign method to it.

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

mainApp.factory('MathService', function() {

   var factory = {};

   factory.multiply = function(a, b) {

      return a * b

   }

   return factory;

});
```

Using Service Method

In this method, we define a service and then assign method to it. We also inject an already available service to it.

```
mainApp.service('CalcService', function(MathService){
```

```
    this.square = function(a) {
              return MathService.multiply(a,a);
      }
});
```

Example

The following example shows use of all the above mentioned directives:

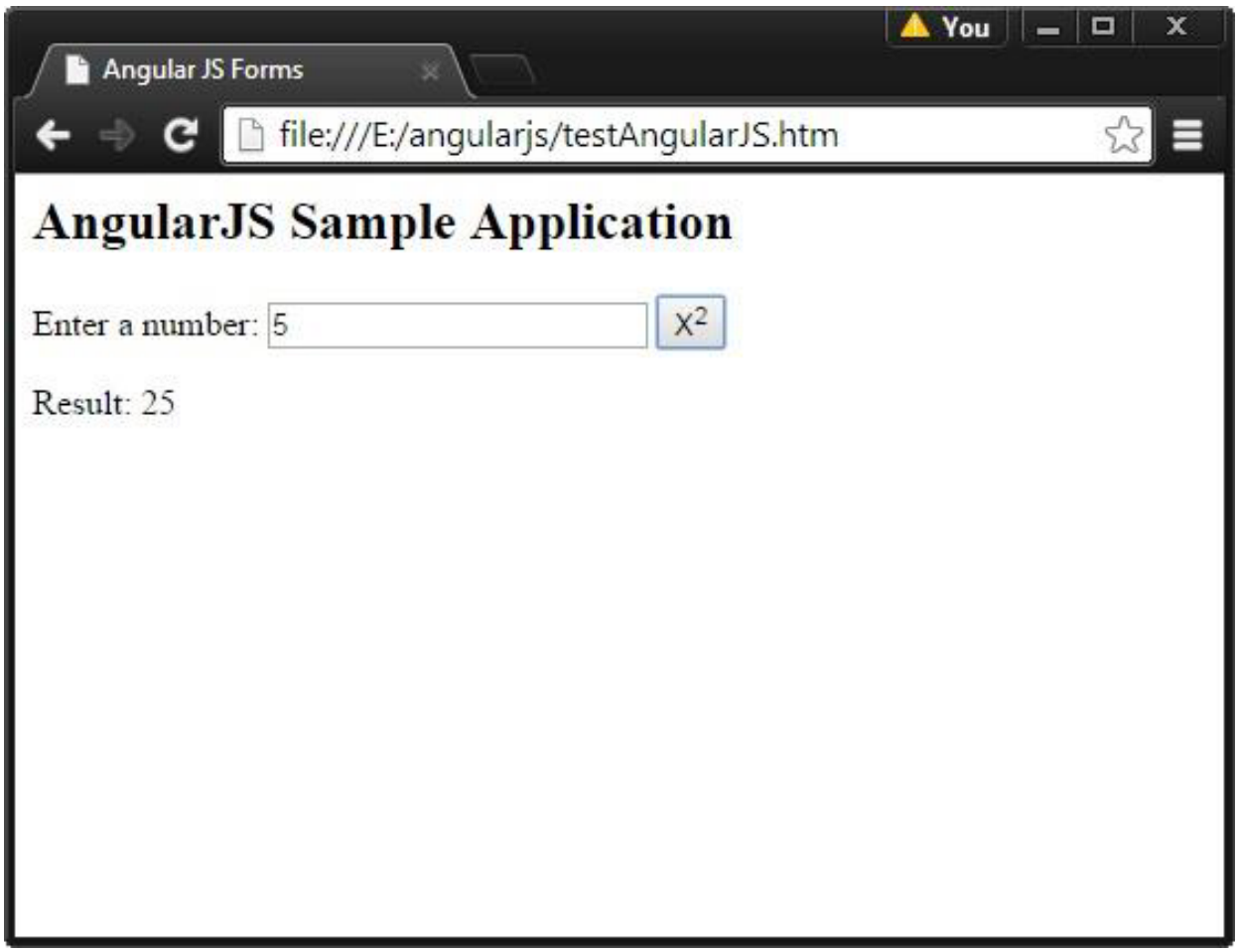
testAngularJS.htm

```
<html>
<head>
   <title>Angular JS Forms</title>
</head>
<body>
   <h2>AngularJS Sample Application</h2>
   <div ng-app="mainApp" ng-controller="CalcController">
      <p>Enter a number: <input type="number" ng-model="number" />
      <button ng-click="square()">X<sup>2</sup></button>
      <p>Result: {{result}}</p>
   </div>
   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>
   <script>
      var mainApp = angular.module("mainApp", []);
      mainApp.factory('MathService', function() {
         var factory = {};
         factory.multiply = function(a, b) {
            return a * b
```

```
         }
         return factory;
      });

      mainApp.service('CalcService', function(MathService){
            this.square = function(a) {
            return MathService.multiply(a,a);
         }
      });

      mainApp.controller('CalcController', function($scope, CalcService) {
            $scope.square = function() {
            $scope.result = CalcService.square($scope.number);
         }
      });
   </script>
</body>
</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.


You
Angular JS Forms
file:///E:/angularjs/testAngularJS.htm
AngularJS Sample Application
Enter a number: 5
X2
Result: 25

18. DEPENDENCY INJECTION

Dependency Injection is a software design in which components are given their dependencies instead of hard coding them within the component. This relieves a component from locating the dependency and makes dependencies configurable. This also helps in making components reusable, maintainable and testable.

AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

- Value
- Factory
- Service
- Provider
- Constant

Value

Value is a simple JavaScript object, which is required to pass values to the controller during config phase (config phase is when AngularJS bootstraps itself).

```
//define a module

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

//create a value object as "defaultInput" and pass it a data.

mainApp.value("defaultInput", 5);

...

//inject the value in the controller using its name "defaultInput"

mainApp.controller('CalcController', function($scope, CalcService,
defaultInput) {

      $scope.number = defaultInput;

      $scope.result = CalcService.square($scope.number);



      $scope.square = function() {

      $scope.result = CalcService.square($scope.number);
```

```
   }

});
```

Factory

Factory is a function which is used to return value. It creates a value on demand whenever a service or a controller requires it. It generally uses a factory function to calculate and return the value.

```
//define a module

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

//create a factory "MathService" which provides a method multiply to return
multiplication of two numbers

mainApp.factory('MathService', function() {

   var factory = {};

   factory.multiply = function(a, b) {

      return a * b

   }

   return factory;

});



//inject the factory "MathService" in a service to utilize the multiply
method of factory.

mainApp.service('CalcService', function(MathService){

      this.square = function(a) {

      return MathService.multiply(a,a);

   }

});

...
```

Service

Service is a singleton JavaScript object containing a set of functions to perform certain tasks. Service is defined using service() function and it is then injected into the controllers.

```
//define a module

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

...

//create a service which defines a method square to return square of a
number.

mainApp.service('CalcService', function(MathService){

      this.square = function(a) {

      return MathService.multiply(a,a);

   }

});

//inject the service "CalcService" into the controller

mainApp.controller('CalcController', function($scope, CalcService,
defaultInput) {

      $scope.number = defaultInput;

      $scope.result = CalcService.square($scope.number);



      $scope.square = function() {

      $scope.result = CalcService.square($scope.number);

   }

});
```

Provider

Provider is used by AngularJS internally to create services, factory etc. during the config phase. The following script can be used to create MathService that we created earlier. Provider is a special factory method with get() method which is used to return the value/service/factory.

```
//define a module

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

...

//create a service using provider which defines a method square to return
square of a number.

mainApp.config(function($provide) {

   $provide.provider('MathService', function() {

      this.$get = function() {

         var factory = {};

         factory.multiply = function(a, b) {

            return a * b;

         }

         return factory;

      };

   });

});
```

Constant

Constants are used to pass values at the config phase considering the fact that value cannot be used during the config phase.

```
mainApp.constant("configParam", "constant value");
```

Example

The following example shows use of all the above mentioned directives:

testAngularJS.htm

```
<html>

<head>

   <title>AngularJS Dependency Injection</title>
```

```
</head>

<body>

   <h2>AngularJS Sample Application</h2>

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

      <p>Enter a number: <input type="number" ng-model="number" />

      <button ng-click="square()">X<sup>2</sup></button>

      <p>Result: {{result}}</p>

   </div>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

   <script>

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


      mainApp.config(function($provide) {

         $provide.provider('MathService', function() {

            this.$get = function() {

               var factory = {};

               factory.multiply = function(a, b) {

                  return a * b;

               }

               return factory;

            };

         });

      });


      mainApp.value("defaultInput", 5);
```

```
      mainApp.factory('MathService', function() {
         var factory = {};
         factory.multiply = function(a, b) {
            return a * b;
         }
         return factory;
      });

      mainApp.service('CalcService', function(MathService){
            this.square = function(a) {
            return MathService.multiply(a,a);
         }
      });

      mainApp.controller('CalcController', function($scope, CalcService,
defaultInput) {
            $scope.number = defaultInput;
            $scope.result = CalcService.square($scope.number);

            $scope.square = function() {
            $scope.result = CalcService.square($scope.number);
         }
      });
   </script>
</body>
</html>
```

Output

Open *testAngularJS.htm* in a web browser and see the result.

19. CUSTOM DIRECTIVES

Custom directives are used in AngularJS to extend the functionality of HTML. They are defined using *directive* function. A custom directive simply replaces the element for which it is activated. During bootstrap, the AngularJS application finds matching elements and does one-time activity using its compile() method of the custom directive. Then it processes the element using link() method of the custom directive based on the scope of the directive. AngularJS provides support to create custom directives for the following elements:

- **Element directive** - This activates when a matching element is encountered.
- **Attribute** - This activates when a matching attribute is encountered.
- **CSS** - This activates when a matching CSS style is encountered.
- **Comment** - This activates when a matching comment is encountered.

Understanding Custom Directive

Define custom HTML tags.

```
<student name="Mahesh"></student><br/>

<student name="Piyush"></student>
```

Define custom directive to handle the above HTML tags.

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



//Create a directive, first parameter is the html element to be attached.

//We are attaching student html tag.

//This directive will be activated as soon as any student element is
encountered in html

mainApp.directive('student', function() {

   //define the directive object

   var directive = {};

   //restrict = E, signifies that directive is Element directive

   directive.restrict = 'E';
```

```
   //template replaces the complete element with its text.

   directive.template = "Student: <b>{{student.name}}</b> , Roll No:
<b>{{student.rollno}}</b>";

   //scope is used to distinguish each student element based on criteria.

   directive.scope = {

       student : "=name"

   }

   //compile is called during application initialization. AngularJS calls
it once when html page is loaded.

   directive.compile = function(element, attributes) {

      element.css("border", "1px solid #cccccc");

        //linkFunction is linked with each element with scope to get the
element specific data.

      var linkFunction = function($scope, element, attributes) {

          element.html("Student: <b>"+$scope.student.name +"</b> , Roll No:
<b>"+$scope.student.rollno+"</b><br/>");

          element.css("background-color", "#ff00ff");

      }

      return linkFunction;

   }

   return directive;

});
```

Define controller to update the scope for directive. Here, we use the value of name attribute as scope's child.

```
mainApp.controller('StudentController', function($scope) {

      $scope.Mahesh = {};

      $scope.Mahesh.name = "Mahesh Parashar";

      $scope.Mahesh.rollno  = 1;
```

```
      $scope.Piyush = {};

      $scope.Piyush.name = "Piyush Parashar";

      $scope.Piyush.rollno  = 2;

});
```

Example

```
<html>

<head>

   <title>Angular JS Custom Directives</title>

</head>

<body>

   <h2>AngularJS Sample Application</h2>

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

               <student name="Mahesh"></student><br/>

               <student name="Piyush"></student>

   </div>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

   <script>

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



      mainApp.directive('student', function() {

         var directive = {};

         directive.restrict = 'E';

         directive.template = "Student: <b>{{student.name}}</b> , Roll No:
<b>{{student.rollno}}</b>";
```

```
         directive.scope = {

            student : "=name"

         }


         directive.compile = function(element, attributes) {

            element.css("border", "1px solid #cccccc");


            var linkFunction = function($scope, element, attributes) {

               element.html("Student: <b>"+$scope.student.name +"</b> ,
Roll No: <b>"+$scope.student.rollno+"</b><br/>");

               element.css("background-color", "#ff00ff");

            }


            return linkFunction;

         }


         return directive;

      });


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

            $scope.Mahesh = {};

            $scope.Mahesh.name = "Mahesh Parashar";

            $scope.Mahesh.rollno  = 1;


            $scope.Piyush = {};

            $scope.Piyush.name = "Piyush Parashar";

            $scope.Piyush.rollno  = 2;

      });
```

```
   </script>
</body>
</html>
```

Output

Open *testAngularJS.htm* in a web browser and see the result.

20. INTERNALIZATION

AngularJS supports inbuilt internationalization for three types of filters : Currency, Date, and Numbers. We only need to incorporate corresponding java script according to locale of the country. By default, it considers the locale of the browser. For example, for Danish locale, use the following script:

```
<script src="https://code.angularjs.org/1.2.5/i18n/angular-locale_da-
dk.js"></script>
```

Example Using Danish Locale

testAngularJS.htm

```
<html>

<head>

   <title>Angular JS Forms</title>

</head>

<body>

   <h2>AngularJS Sample Application</h2>

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

      {{fees | currency }}  <br/><br/>

      {{admissiondate | date }}   <br/><br/>

      {{rollno | number }}

   </div>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

   <script src="https://code.angularjs.org/1.2.5/i18n/angular-locale_da-
dk.js"></script>

   <script>

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

```
      mainApp.controller('StudentController', function($scope) {
            $scope.fees = 100;
                  $scope.admissiondate  = new Date();
            $scope.rollno = 123.45;
      });

   </script>
</body>
</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.

Example Using Browser Locale

testAngularJS.htm

```
<html>
<head>
   <title>Angular JS Forms</title>
```

```
</head>

<body>

   <h2>AngularJS Sample Application</h2>

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

      {{fees | currency }}  <br/><br/>

      {{admissiondate | date }}   <br/><br/>

      {{rollno | number }}

   </div>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

   <!-- <script src="https://code.angularjs.org/1.2.5/i18n/angular-
locale_da-dk.js"></script> -->

   <script>

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


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

            $scope.fees = 100;

                     $scope.admissiondate  = new Date();

            $scope.rollno = 123.45;

      });


   </script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.


You
Angular JS Forms
file:///E:/angularjs/testAngularJS.htm
AngularJS Sample Application
$100.00
Oct 16, 2014
123.45