AngularJS

Introduction AngularJS

AngularJS is a JavaScript framework developed by Google for building dynamic web applications.

It extends HTML with additional attributes and binds data to HTML with expressions.

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

It was originally developed in 2009 by Misko Hevery and Adam Abrons.

Advantages of AngularJS

The advantages of AngularJS are:

- AngularJS provides the capability to create Single Page Application in a very clean and maintainable way.
- AngularJS provides data binding capability to HTML. Thus, it gives users a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and makes use of separation of concerns.
- AngularJS provides reusable components.

Disadvantages of AngulaJS

Not Secure : Being JavaScript only framework, applications 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

What are Directives?

- **Directives** are markers on DOM elements that tell AngularJS's HTML compiler (**`$compile`**) to attach a specified behaviour to the DOM element or transform the DOM element and its children.
- They are a core feature of AngularJS that enables the creation of custom, reusable HTML components.

Commonly Used Directives:

- **ng-model:** Binds HTML controls (like input, select, textarea) to application data using two-way data binding.
- **ng-repeat**: Iterates over a collection and instantiates a template for each item in the collection.
- **ng-show / ng-hide:** Shows or hides an element based on a boolean expression.
- **ng-if:** Conditionally renders an element based on a boolean expression.
- **ng-click**: Attaches a click event handler to an element.
- **ng-class**: Dynamically adds or removes CSS classes based on conditions.

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>
```

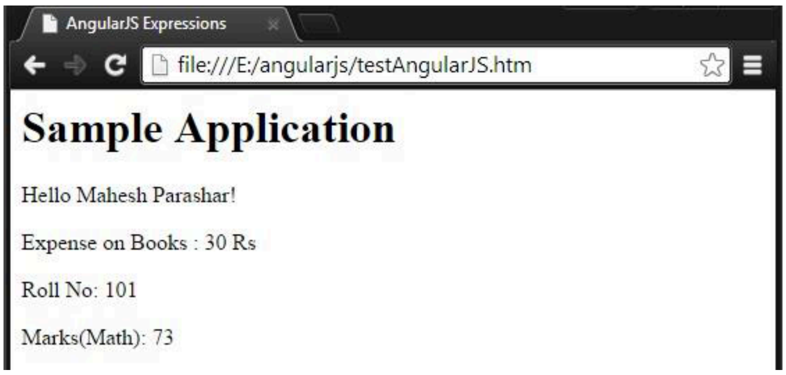
Using String
```
<p>Hello {{student.firstname + " " + student.lastname}}!</p>
```

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

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

Example : testAngularJS.html
The following example shows use of all the above mentioned expressions:

```
<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>
```



Sample Application

Hello Mahesh Parashar!

Expense on Books : 30 Rs

Roll No: 101

Marks(Math): 73

Controller

What are Controllers?

- Controllers in AngularJS are JavaScript functions that are used to augment the AngularJS scope.
- They are responsible for defining the behavior of a particular section of the UI.
- Controllers are defined as part of an AngularJS module using the `controller()` method.
- The controller function takes a name and a function as arguments.

Purpose of Controllers:

- Controllers are used to initialize the data model and add behavior to the scope.
- They act as a glue between the HTML view and the data model, facilitating dynamic content rendering.

Example of Controller:

```
angular.module('myApp').controller('MyController', function($scope) {
   $scope.message = 'Hello, AngularJS!';
});
```

Binding Data:

- Controllers initialize the data model by attaching properties and functions to the scope.
- Data binding in AngularJS allows these properties to be automatically synchronized between the controller and the view.

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.

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

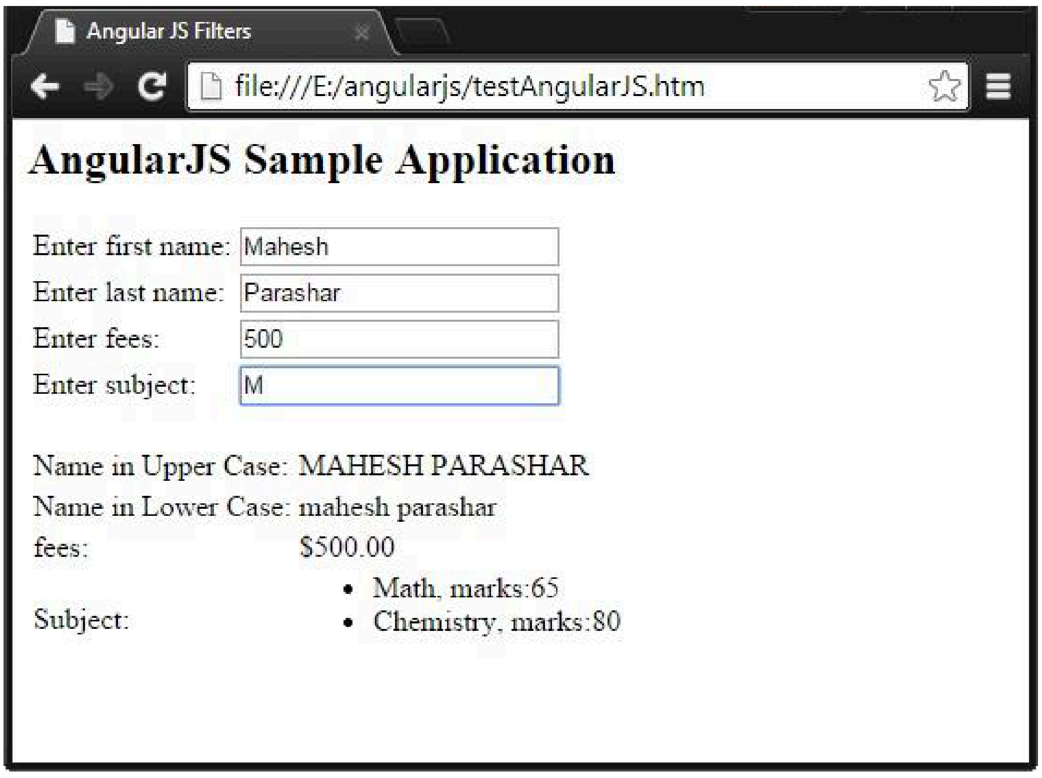
To order subjects by marks, we use orderBy marks.
Subject:
```
<ul>
<li ng-repeat="subject in student.subjects | orderBy:'marks'">
{{ subject.name + ', marks:' + subject.marks }}
</li>
</ul>
```

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.



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
                      • Math, marks:65
Subject:              • Chemistry, marks:80
```

};

services

AngularJS services are objects or functions that are reusable across an application and can be injected into components such as controllers, directives, filters, and other services. They provide a way to organize and share code, facilitate modularity, and promote the separation of concerns in AngularJS applications. Here's a brief overview of AngularJS services:

1. Types of Services:

- Built-in Services: AngularJS comes with several built-in services such as `$http` for making AJAX requests, `$q` for promises, `$timeout` for timers, `$location` for URL handling, etc.
- Custom Services: Developers can create custom services to encapsulate application-specific logic, data access, business logic, and utility functions.

2. Characteristics of Services:

- Singleton: AngularJS services are singleton objects, meaning that only one instance of a service is created and shared across the entire application.
- Lazy Instantiation: Services are lazily instantiated, meaning they are only created when they are first requested and are then cached for subsequent use.
- Injectable: Services can be injected into other components (controllers, directives, filters, etc.) using AngularJS's dependency injection mechanism.

3. Creating Custom Services:

- Custom services are created using the `service()` or `factory()` methods of AngularJS modules.
- The `service()` method is used to define services using constructor functions, while the `factory()` method allows for more flexibility in defining services.

Example of Custom Service:

```
angular.module('myApp').service('userService', function() {

  var users = ['John', 'Jane', 'Doe'];

  this.getUsers = function() {

    return users;

  };

  this.addUser = function(user) {
        users.push(user);

 };

});
```

In this example, a custom service named `userService` is defined to manage user-related functionality.

4. Injecting Services:

- Services can be injected into other components by specifying their names as dependencies in the constructor function.
- AngularJS's dependency injection system takes care of resolving and providing the required services to components.

5. Usage in Controllers:

```
angular.module('myApp').controller('UserController', function($scope, userService) {

   $scope.users = userService.getUsers();

   $scope.addUser = function(user) {

      userService.addUser(user);

   };

});
```

In this example, the `userService` is injected into the `UserController` to fetch and manage user data.

6. Benefits of Services:

- Promote code reusability and maintainability by encapsulating common functionality.
- Facilitate modularity and separation of concerns by abstracting complex logic into reusable components.
- Enable better testability through dependency injection and mocking.

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:

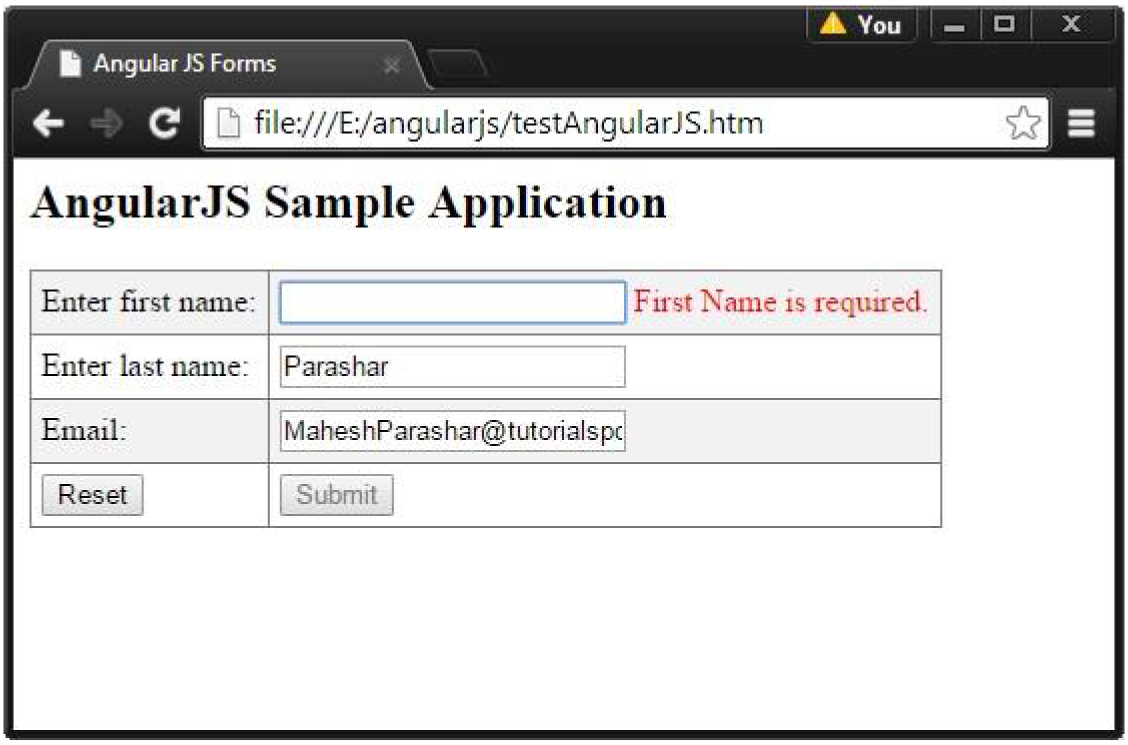
- **ng-click**
- **ng-dbl-click**
- **ng-mousedown**
- **ng-mouseup**
- **ng-mouseenter**
- **ng-mouseleave**
- **ng-mousemove**
- **ng-mouseover**
- **ng-keydown**
- **ng-keyup**
- **ng-keypress**
- **ng-change**

ng-click

Reset data of a form using the 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> 12. FORMS

<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>
```

output:



AngularJS Sample Application

Enter first name:	First Name is required.
Enter last name:	Parashar
Email:	MaheshParashar@tutorialspc
Reset	Submit

AngularJS Forms:

AngularJS provides a powerful and flexible mechanism for working with forms in web applications.
Forms in AngularJS are built using the ngForm directive and various input directives.

Key Features of AngularJS Forms:

Two-Way Data Binding: AngularJS forms facilitate two-way data binding between form controls and model data.
Validation: AngularJS provides built-in validation directives and services for form validation, including both client-side and server-side validation.
Error Handling: AngularJS forms automatically handle error states and provide mechanisms for displaying error messages to users.
Form Submission: AngularJS allows you to easily handle form submission and perform actions such as sending data to a server.

Creating Forms:

Forms in AngularJS are created using the <form> element along with the ngForm directive.
Form controls are added using various input directives such as <input>, <select>, <textarea>, etc.

Form Validation:

AngularJS provides a set of built-in validation directives such as ngRequired, ngMinlength, ngMaxlength, ngPattern, etc.
Custom validation logic can be implemented using custom validation directives and validation functions.

Example Form Markup:

```
<form name="myForm" ng-submit="submitForm()">
   <input type="text" name="username" ng-model="user.username" ng-minlength="3"
ng-maxlength="10" required>
   <div ng-messages="myForm.username.$error" ng-show="myForm.username.$touched">
      <div ng-message="required">Username is required.</div>
      <div ng-message="minlength">Username must be at least 3 characters long.</div>
      <div ng-message="maxlength">Username cannot exceed 10 characters.</div>
   </div>
   <button type="submit">Submit</button>
```

```
</form>
```

In this example, the form contains an input field for the username with validation messages displayed based on the input state.

Handling Form Submission:

Form submission is handled using the ngSubmit directive, which binds to a function in the controller.
The function specified in ngSubmit is executed when the form is submitted.