What is AngularJS?

- AngularJS is a JavaScript framework. It is a library written in JavaScript.
- It provides data-binding, basic templating directives, form validation, routing, deep-linking, reusable components, dependency injection.
- It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly.
- Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write.
- AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions.
- Angular was built with the CRUD application in mind.
- Angular provides the following enhancement to HTML through directive:

  - ✓ Data binding, as in {{}}.
  - ✓ DOM control structures for repeating/hiding DOM fragments.
  - ✓ Support for forms and form validation.
  - ✓ Attaching new behavior to DOM elements, such as DOM event handling.
  - ✓ Grouping of HTML into reusable components.

Angular Handles

- **Registering callbacks**: Registering callbacks clutters your code, making it hard to see the forest for the trees. It vastly reduces the amount of JavaScript coding you have to do, and it makes it easier to see what your application does.
- **Manipulating HTML DOM programmatically**: Manipulating HTML DOM is a cornerstone of AJAX applications, but it's cumbersome and error-prone. By declaratively describing how the UI should change as your application state changes, you are freed from low-level DOM manipulation tasks. Most applications written with Angular never have to programmatically manipulate the DOM, although you can if you want to.
- **Marshaling data to and from the UI**: CRUD operations make up the majority of AJAX applications' tasks. The flow of marshaling data from the server to an internal object to an HTML form, allowing users to modify the form, validating the form, displaying validation errors,

returning to an internal model, and then back to the server, creates a lot of boilerplate code. Angular eliminates almost all of this boilerplate, leaving code that describes the overall flow of the application rather than all of the implementation details.

- **Angular bootstraps your app**: Allows you to easily use services, which are auto-injected into your application in using dependency-injection.

AngularJS History

- AngularJS is quite new. Version 1.0 was released in **2012**.
- Misko Hevery, a Google employee, started to work on AngularJS in **2009**.
- The idea turned out very good, and the project is now officially backed by a Google development team.

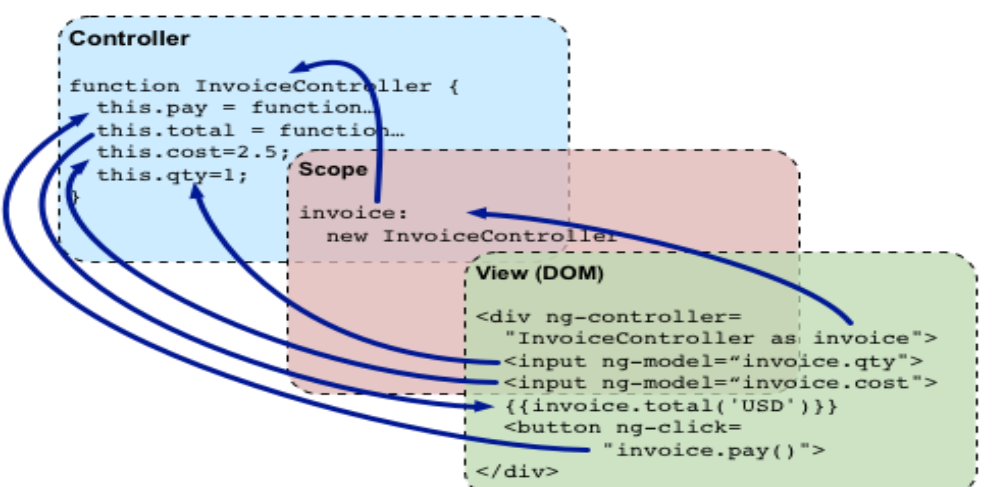
FirstPage.html

```
<!DOCTYPE html>
<html>
<head>
    <title>First Angular App</title>
    <script src="angular.js"></script>
    <script>
        var module = angular.module("first", []);
        module.controller("FirstController",
             function ($scope)    {
              $scope.message = "My First AngularJS Page!";
              $scope.today = new Date().toString();
             }
        );
    </script>
</head>
<body ng-app="first">
    <h1 ng-controller="FirstController">
           {{ message }} <p/> {{today}}
    </h1>

</body>
</html>
```

How is it executed?

- First the HTML document is loaded into the browser, and evaluated by the browser. At this time the AngularJS JavaScript file is loaded, the angular global object is created, and your JavaScript which registers controller functions is executed.
- Second, AngularJS scans through the HTML to look for AngularJS apps and views. When AngularJS finds a view it connects that view to the corresponding controller function.
- Third, AngularJS executes the controller functions and update (render) the views with data from the model populated by the controller. The page is now ready.
- Fourth, AngularJS listens for browser events (e.g. input fields being changed, buttons clicked, the mouse moved etc.). If any of these browser events require a view to change, AngularJS will update the view correspondingly. If the event requires that the corresponding controller function is called, AngularJS will do that too. Not all events require the controller function to be called, even if the view is updated as a result of the event.
- AngularJS views mix data from the model into an HTML template. You use AngularJS directives to tell AngularJS how to mix the data into the HTML template.



Template

- Template is an HTML page.
- When Angular starts your application, it parses and processes this new markup from the template using the so-called "compiler".
- The loaded, transformed and rendered DOM is then called the "view".
- Angular combines the template with information from the model and controller to render the dynamic view that a user sees in the browser.

These are the types of Angular elements and attributes you can use:

Directive	An attribute or element that augments an existing DOM element or represents a reusable DOM component.
Markup	The double curly brace notation {{ }} to bind expressions to elements is built in Angular markup.
Filter	Formats data for display.
Form controls	Validates user input.

AngularJS Expressions

Angular expressions are JavaScript-like code snippets that are mainly placed in interpolation bindings such as below:

```
<span title="{{attrBinding}}"> {{textBinding }} </span>
```

Expressions are also used directly in directive attributes as follows:

```
ng-click="functionExpression()"
```

For example, these are valid expressions in Angular:

- 1+2
- a+b
- user.name
- items[index]

Angular expressions do not have direct access to global variables like window, document or location. This restriction is intentional.

Instead use services like $**window** and $**location** in functions on controllers, which are then called from expressions.

Angular Expressions vs. JavaScript Expressions

Angular expressions are like JavaScript expressions with the following differences:

- **Context**: JavaScript expressions are evaluated against the global window. In Angular, expressions are evaluated against a scope object.
- **Forgiving**: In JavaScript, trying to evaluate undefined properties generates ReferenceError or TypeError. In Angular, expression evaluation is forgiving to undefined and null.
- **No Control Flow Statements**: You cannot use the following in an Angular expression: conditionals, loops, or exceptions. Apart from the ternary operator (a ? b : c), you cannot write a control flow statement in an expression. The reason behind this is core to the Angular philosophy that application logic should be in controllers, not the views. If you need a real conditional, loop, or to throw from a view expression, delegate to a JavaScript method instead.
- **No Function Declarations**: You cannot declare functions in an Angular expression, even inside ng-init directive.

Note: If you want to run more complex JavaScript code, you should make it a controller method and call the method from your view. If you want to `eval()` an Angular expression yourself, use the `$eval()` method.

One-time Binding

An expression that starts with **::** is considered a one-time expression. One-time expressions will stop recalculating once they are stable, which happens after the first digest if the expression results is a non-undefined value.

```
<p id="one-time-binding-example">One time binding:
{{::name}}</p>
```

Interpolation and data-binding

Interpolation markup with embedded expressions is used by Angular to provide data-binding to text nodes and attribute values.

```
<a ng-href="img/{{username}}.jpg">Hello {{username}}!</a>
```

During the compilation process the compiler uses the $interpolate service to see if text nodes and element attributes contain interpolation markup with embedded expressions.

If that is the case, the compiler adds an interpolateDirective to the node and registers watches on the computed interpolation function, which will update the corresponding text nodes or attribute values as part of the normal digest cycle.

Angular provides special ng-prefixed directives for the following boolean attributes: disabled, required, selected, checked, readOnly , and open.

These directives take an expression inside the attribute, and set the corresponding boolean attribute to true when the expression evaluates to true.

```
Disabled: <input type="checkbox" ng-model="isDisabled" />
<button ng-disabled="isDisabled">Disabled</button>
```

If an attribute with a binding is prefixed with the `ngAttr` prefix (denormalized as `ng-attr-`) then during the binding it will be applied to the corresponding unprefixed attribute.

```
<circle ng-attr-cx="{{cx}}"></circle>
```

Angular directive attributes take either expressions or interpolation markup with embedded expressions. It is considered bad practice to embed interpolation markup inside an expression.

Directives

- The first kind of new markup are the so-called "directives".
- They apply special behavior to attributes or elements in the HTML.
- Angular also defines a directive for the input element that adds extra behavior to the element.
- The ng-model directive stores/updates the value of the input field into/from a variable.

Directive	Description
ng-app	Defines the root element of an application.
ng-bind	Binds the innerHTML of HTML elements to application data. An alternative to {{ }}
ng-controller	Defines the controller object for an application.
ng-disabled	Binds application data to the HTML disabled attribute.
ng-hide	Hides or shows HTML elements.
ng-include	Includes HTML in an application.
ng-init	Defines initial values for an application.
ng-model	Binds the value of HTML controls to application data.
ng-repeat	Defines a template for each data in a collection.
ng-show	Shows or hides HTML elements.
ng-non-bindable	Tells Angular not to compile or bind the contents of the current DOM element.
ng-cloak	Prevents HTML template from being displayed in raw form while application is being loaded.
ng-href	Used to specify href for anchor elements.
ng-src	Used to specify source for image elements.
ng-if	Adds or removes the associated DOM tree.
ng-switch	Chooses one of the nested elements and makes it visible based on which element matches the value obtained from the evaluated expression

```
<ANY ng-switch="expression">
  <ANY ng-switch-when="matchValue1">...</ANY>
  <ANY ng-switch-when="matchValue2">...</ANY>
  <ANY ng-switch-default>...</ANY>
</ANY>
```

ngRepeat Directive

The ngRepeat directive instantiates a template once per item from a collection. Each template instance gets its own scope, where the given loop variable is set to the current collection item, and $index is set to the item index or key.

```
<ng-repeat
  ng-repeat="repeat_expression">
...
</ng-repeat>
```

```
<ANY  ng-repeat="repeat_expression">
...
</ANY>
```

Repeat Expression can be any of the following:

variable in expression
Where variable is the user defined loop variable and expression is a scope expression giving the collection to enumerate.

```
album in artist.albums.
```

(key, value) in expression
Where key and value can be any user defined identifiers, and expression is the scope expression giving the collection to enumerate.

```
(name, age) in {'Java':22, 'C#':15}.
```

Special properties are exposed on the local scope of each template instance, including:

Variable	Type	Details
$index	number	iterator offset of the repeated element (0..length-1)
$first	boolean	true if the repeated element is first in the iterator.
$middle	boolean	true if the repeated element is between the first and last in the iterator.

$last	boolean	true if the repeated element is last in the iterator.
$even	boolean	true if the iterator position $index is even (otherwise false).
$odd	boolean	true if the iterator position $index is odd (otherwise false).

Directives ng-class, ng-class-even, ng-class-odd

The ngClass directive allows you to dynamically set CSS classes on an HTML element by data binding an expression that represents all classes to be added.

The directive operates in three different ways, depending on which of three types the expression evaluates to:

- If the expression evaluates to a string, the string should be one or more space-delimited class names.
- If the expression evaluates to an array, each element of the array should be a string that is one or more space-delimited class names.
- If the expression evaluates to an object, then for each key-value pair of the object with a truthy value the corresponding key is used as a class name.

```
<p ng-class="{strike: deleted, bold: important, red:
error}">Map Syntax Example</p>
<input type="checkbox" ng-model="deleted"> deleted (apply
"strike" class)<br>
<input type="checkbox" ng-model="important"> important (apply
"bold" class)<br>
<input type="checkbox" ng-model="error"> error (apply "red"
class)<hr>
<p ng-class="style">Using String Syntax</p>
<input type="text" ng-model="style" placeholder="Type: bold
strike red">
```

The **ng-class-odd** and **ng-class-even** directives work exactly as **ng-class**, except they work in conjunction with **ng-repeat** and take effect only on odd (even) rows.

Scope

- The concept of a scope in Angular is crucial.
- A scope can be seen as the glue which allows the template, model and controller to work together.
- Angular uses scopes, along with the information contained in the template, data model, and controller, to keep models and views separate, but in sync.
- Any changes made to the model are reflected in the view; any changes that occur in the view are reflected in the model.
- Scopes are arranged in hierarchical structure
- Scopes can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either "child scopes" or "isolate scopes"
- The $watch allows the directives to be notified of property changes, which allows the directive to render the updated value to the DOM.
- The location where the root scope is attached to the DOM is defined by the location of ng-app directive.

Controller

- In Angular, a Controller is a JavaScript constructor function that is used to augment the Angular Scope.
- When a Controller is attached to the DOM via the **ng-controller** directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function.
- A new child scope will be available as an injectable parameter to the Controller's constructor function as $scope.
- Controller is used to Set up the initial state of the $scope object and add behavior to the $scope object.
- Controller should contain only business logic.
- AngularJS controllers are regular JavaScript Objects
- A controller can also have methods (functions as object properties).

```
<div ng-app="" ng-controller="personController">
First Name: <input type="text" ng-model="firstName"><br/>
Last Name: <input type="text" ng-model="lastName"><br/>
Full Name: {{fullName()}}</div>
```

```
<script>
function personController($scope) {
    $scope.firstName = "Srikanth";
    $scope.lastName = "Pragada";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    }
 }
</script>
```

Modules

- A module is a container for the different parts of your app – controllers, services, filters, directives, etc.
- Modules make your application more readable, and keep the global namespace clean.
- Global values should be avoided in applications. They can easily be overwritten or destroyed by other scripts. So modules help us to keep our components away from global namespace.
- AngularJS recommend that you break your application to multiple modules like this:

  - ✓ A module for each feature
  - ✓ A module for each reusable component (especially directives and filters)
  - ✓ And an application level module which depends on the above modules and contains any initialization code.

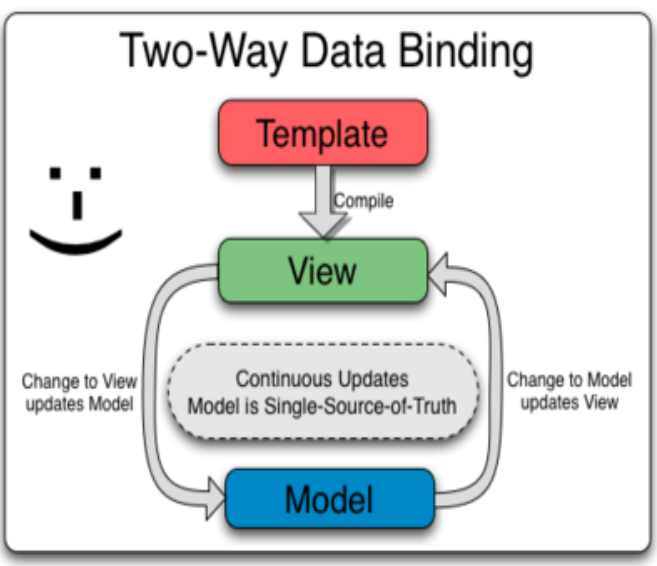
$scope Object

- Typically, when you create an application you need to set up the initial state for the Angular $scope.
- You set up the initial state of a scope by attaching properties to the $scope object. The properties contain the view model (the model that will be presented by the view).
- All the $scope properties will be available to the template at the point in the DOM where the Controller is registered.
- In order to react to events or execute computation in the view we must provide behavior to the scope. We add behavior to the scope by

attaching methods to the $scope object. These methods are then available to be called from the template/view.

Data Binding

- Data-binding in Angular apps is the automatic synchronization of data between the model and view components.
- The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.
- Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it.



AngularJS Filters

- Filters can be added to expressions and directives using a pipe character.
- It is possible to chain filters by simply putting more filters after each other in the filter section. When chaining filters, the output of one filter is used as input for the next filter in the chain.

```
{{ expression | filter:argument1:argument2:... }}
```

```
<p>The name is {{ lastName | uppercase }}</p>
```

Filter	Description
currency	Format a number to a currency format.
filter	Select a subset of items from an array.
lowercase	Format a string to lower case.
uppercase	Format a string to upper case.
date	Formats the date according to the given format.
number	Formats the value as number
json	Converts the value to JSON string

```
{{ course.Price | currency : 'Rs ' : 2}}
{{ course.StartDate | date :"dd MMM yyyy" }}
{{ course | json }}
```

Array Filters

AngularJS also contains a set of array filters which filters or transforms arrays.

Filter	Description
limitTo	Limits the array to the given size, beginning from some index in the array. The limitTo filter also works on strings.
filter	A general purpose filter.
orderBy	Sorts the array based on provided criteria.

```
<div ng-repeat="n in names | orderBy :  '-' ">
       {{ n }}
</div>
```

```
<tr ng-repeat="c in courses | orderBy:'duration' : 'reverse'">
       <td> {{ c.name }}</td>
       <td> {{ c.duration }}</td>
</tr>
```

```
$scope.longCourses = function (c, idx) {
       return c.duration > 50;
};
```

```
<tr ng-repeat="c in courses | filter: longCourses | orderBy :
'duration' : 'reverse' ">
                    <td> {{ c.name }}</td>
                   <td> {{ c.duration }}</td>
</tr>
```

Data Types

- AngularJS numbers are like JavaScript numbers.
- AngularJS strings are like JavaScript strings.
- AngularJS objects are like JavaScript objects.
- AngularJS arrays are like JavaScript arrays

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">
  <p>The points are {{ points[2] }}</p>
</div>
```

AngularJS Event Listener Directives

- You attach an event listener to an HTML element using one of the AngularJS event listener directives:

  - ✓ 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

Angular Forms

- A Form is a collection of controls for the purpose of grouping related controls together.
- Form and controls provide validation services, so that the user can be notified of invalid input.
- Attribute `novalidate` is used to disable browser's native form validation.
- The value of `ngModel` won't be set unless it passes validation for the input field. For example: inputs of type `email` must have a value in the form of `user@domain`.
- To allow styling of form as well as controls, `ngModel` adds these CSS classes:

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

- ✓ ng-pending: any $asyncValidators are unfulfilled

```
<input type="checkbox" ng-model="myForm.wantNewsletter"
ng-true-value="yes" ng-false-value="no">
<input type="radio" ng-model="myForm.whichNewsletter"
value="weeklyNews">
<input type="radio" ng-model="myForm.whichNewsletter"
value="monthlyNews">
```

ng-options

Instead of using static HTML options you can have AngularJS create option elements based on data from the $scope object.

```
<select ng-model="myForm.car"
   ng-options="obj.id as obj.name for obj in myForm.options">
      <option value="">Please choose a car</option>
</select>
```

Form Validation

- ❑ AngularJS has a set of form validation directives you can use.
- ❑ AngularJS validates form fields before copying their value into the $scope properties to which the form fields are bound.
- ❑ If a form field is invalid, its value is NOT copied into the $scope property it is bound to.
- ❑ Instead the corresponding $scope property is cleared. That is done to prevent the $scope properties from containing invalid values.
- ❑ The ng-minlength and ng-maxlength form validation directives can be used to validate the length of data entered in a form field.
- ❑ The ng-pattern directive can be used to validate the value of an input field against a regular expression.
- ❑ ng-required The ng-required directive checks if the value of the form field is empty or not. Actually, you just use the required attribute of HTML5, and AngularJS detects it automatically.

Checking Field Validation State

- If you give the <form> element a name attribute, then the form will be added to the $scope object as a property.
- Each form directive creates an instance of FormController.
- FormController keeps track of all its controls and nested forms as well as the state of them, such as being valid/invalid or dirty/pristine.

```
<form name="myFormNg" ng-submit="myForm.submitTheForm()" >
 ...
</form>
```

When you call a function on the $scope object (a function added to the $scope object by your controller function), you can access the ngFormController object via its name, like this:

```
$scope.myFormNg
```

Both ngFormController and ngModelController objects contain a set of properties that tell if the form or input field is valid. The properties are:

Property	Description
$pristine	True if the form has not been changed (no form fields has changed), false if some fields have been changed.
$dirty	The reverse of $pristine - false if the form has not been changed - true if it has.
$valid	True if the form field (or the whole form = all form fields) is valid. False if not.
$invalid	The reverse of the $valid - false if the field (or all fields in the form) is valid, true if the field (or a single field in the for) is invalid.
$submitted	True if user has submitted the form even if it is invalid.

Validation Directives

The following are directives related to validating a form input field.

```
<input   ng-model=""  [name=""]
  [required=""]   [ng-required=""]   [ng-minlength=""]
  [ng-maxlength=""]    [ng-pattern=""]    [ng-change=""]
  [ng-trim=""]>
...
</input>
```

Param	Type	Details
ngModel	string	Assignable angular expression to data-bind to.
name (optional)	string	Property name of the form under which the control is published.
required (optional)	string	Sets required validation error key if the value is not entered.
ngRequired (optional)	boolean	Sets required attribute if set to true
ngMinlength (optional)	number	Sets minlength validation error key if the value is shorter than minlength.
ngMaxlength (optional)	number	Sets maxlength validation error key if the value is longer than maxlength. Setting the attribute to a negative or non-numeric value, allows view values of any length.
ngPattern (optional)	string	Sets pattern validation error key if the value does not match the RegExp pattern expression. Expected value is /regexp/ for inline patterns or regexp for patterns defined as scope expressions.
ngChange (optional)	string	Angular expression to be executed when input changes due to user interaction with the input element.
ngTrim (optional)	boolean	If set to false Angular will not automatically trim the input. This parameter is ignored for input[type=password] controls, which will never trim the input. (default: true)

$**error** contains the key that is related to failed validation.

For example if **minlength** validation failed for a field then the following for that field is true:

```
if (form.username.$error.minlength)
            // minlength validation for username failed
```

You can user this in HTML as follows:

```
<span class="error" ng-show="form.username.$error.minlength">
      Too short!</span>
```

Checkbox Control

The following are important attributes of checkbox.

```
<input type="checkbox"
       ng-model=""
       [name=""]
       [ng-true-value=""]
       [ng-false-value=""]
       [ng-change=""]>
```

Date Control

The following are important attributes of Date control.

```
<input type="date"
       ng-model=""
       [name=""]
       [min=""]
       [max=""]
       [required=""]
       [ng-required=""]
       [ng-change=""]>
```