

Index

1. Introduction	2-7
2. Environmental Setup	8-10
3. Directives	11-15
4. Custom Directives	16-22
5. JSON	23-25
6. Services	26-41
7. Validations	42-42
8. Filters	42-43
9. I18N	43-44
10. Programs	45-57
11. NodeJS	58-71
12. Testing	72-79
13. Grunt	80-86
14. Questions & Answers	87-136

AngularJS

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.

Advantages of AngularJS:

- No need to use observable functions; Angular analyses the page DOM and builds the bindings based on the Angular-specific element attributes. That requires less writing, the code is cleaner, easier to understand and less error prone.
- Angular modifies the page DOM directly instead of adding inner HTML code. That is faster.
- Data binding occurs not on each control or value change (no change listeners) but at particular points of the JavaScript code execution. That dramatically improves performance as a single bulk Model/View update replaces hundreds of cascading data change events.
- Quite a number of different ways to do the same things, thus accommodating to particular development styles and tasks.
- Extended features such as dependency injection, routing, animations, view orchestration, and more.
- Supported by IntelliJ IDEA and Visual Studio .NET IDEs.
- Supported by Google and a great development community.

Disadvantages of AngularJS:

- Confusion

There are multiple ways to do the same thing with AngularJS. Sometimes, it can be hard for novices to say which way is better for a task. Hence, it is imperative for programmers to develop an understanding of the various components and how they help.

- Lagging UI

If there are more than 2000 watchers, it can get the UI to severely lag. This means that the possible complexity of Angular Forms is limited. This includes big data grids and lists.

- Name Clashes

With AngularJS, you don't have the ability to compose many NG-apps on the same page. This can cause name clashes.

Key Features of AngularJS:

Data-binding:

It is the automatic synchronization of data between model and view components.

Scope:

These are objects that refer to the model. They act as 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.

AngularJS Expressions:

AngularJS expressions are JavaScript-like code snippets that are mainly placed in interpolation bindings such as `{{ textBinding }}`, but also used directly in directive attributes such as `ng-click="functionExpression()"`.

For example, these are valid expressions in AngularJS:

- `1+2`

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

AngularJS Expressions vs. JavaScript Expressions

AngularJS expressions are like JavaScript expressions with the following differences:

- Context: JavaScript expressions are evaluated against the global window. In AngularJS, expressions are evaluated against a [scope](#) object.
- Forgiving: In JavaScript, trying to evaluate undefined properties generates ReferenceError or TypeError. In AngularJS, expression evaluation is forgiving to undefined and null.
- Filters: You can use [filters](#) within expressions to format data before displaying it.
- No Control Flow Statements: You cannot use the following in an AngularJS expression: conditionals, loops, or exceptions.
- No Function Declarations: You cannot declare functions in an AngularJS expression, even inside ng-init directive.
- No RegExp Creation With Literal Notation: You cannot create regular expressions in an AngularJS expression.
- No Object Creation With New Operator: You cannot use new operator in an AngularJS expression.
- No Bitwise, Comma, And Void Operators: You cannot use [Bitwise](#), , or void operators in an AngularJS expression.

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 AngularJS expression yourself, use the [\\$eval\(\)](#) method.

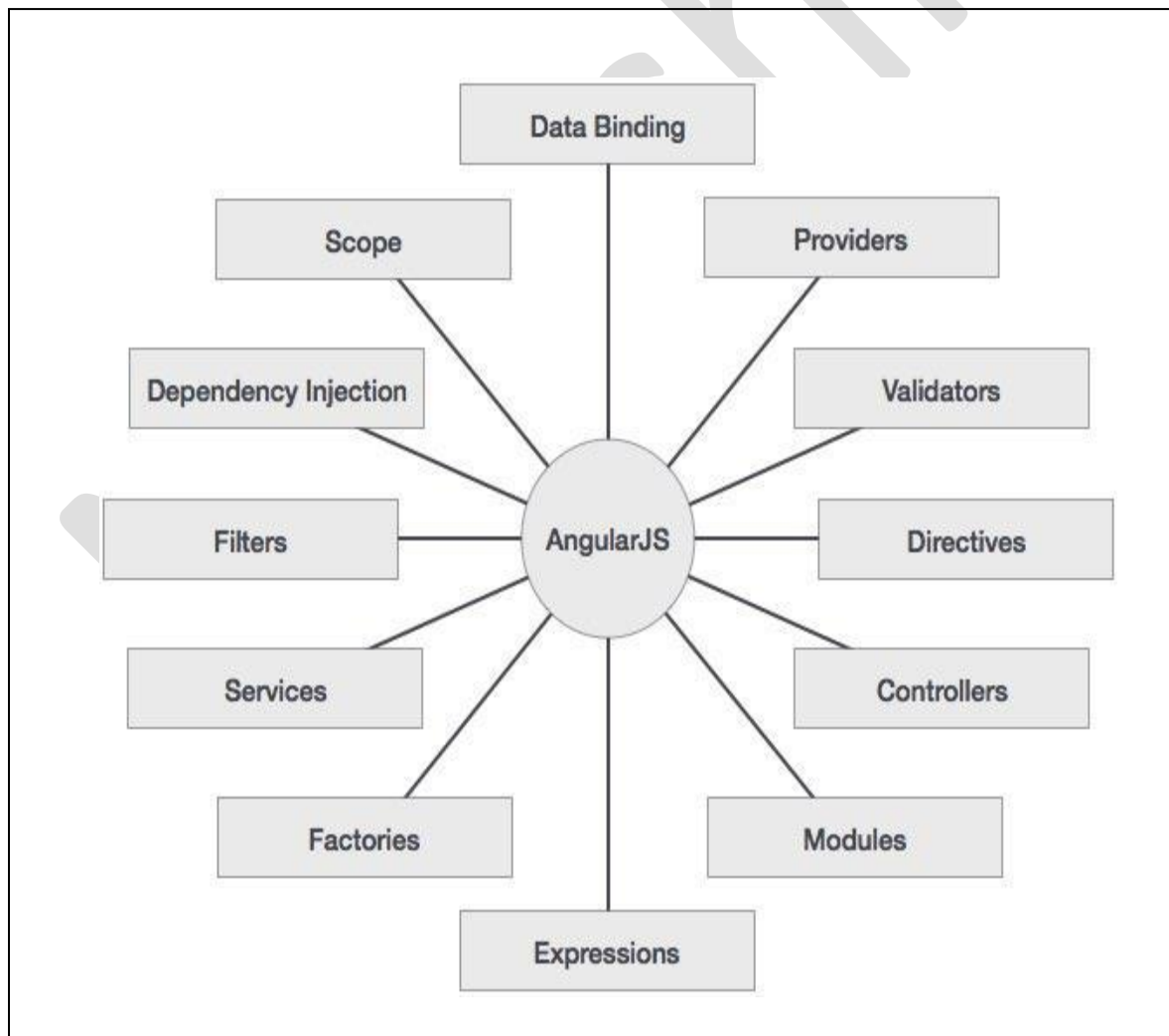
Ex.

```
<span>
  1+2={{1+2}}
</span>
<div ng-controller="ExampleController" class="expressions">
  Expression:
  <input type='text' ng-model="expr" size="80"/>
  <button ng-click="addExp(expr)">Evaluate</button>
  <ul>
    <li ng-repeat="expr in exprs track by $index">
      [ <a href="" ng-click="removeExp($index)">X</a> ]
      <code>{{expr}}</code> => <span ng-bind="$parent.$eval(expr)"></span>
    </li>
  </ul>
</div>
```

Context:

- AngularJS does not use JavaScript's `eval()` to evaluate expressions.
- Instead AngularJS's [\\$parse](#) service processes these expressions.
- AngularJS expressions do not have direct access to global variables like `window`, `document` or `location`. This restriction is intentional.
- It prevents accidental access to the global state – a common source of subtle bugs.
- Instead use services like `$window` and `$location` in functions on controllers, which are then called from expressions. Such services provide mockable access to globals.
- It is possible to access the context object using the identifier `this` and the locals object using the identifier `$locals`.

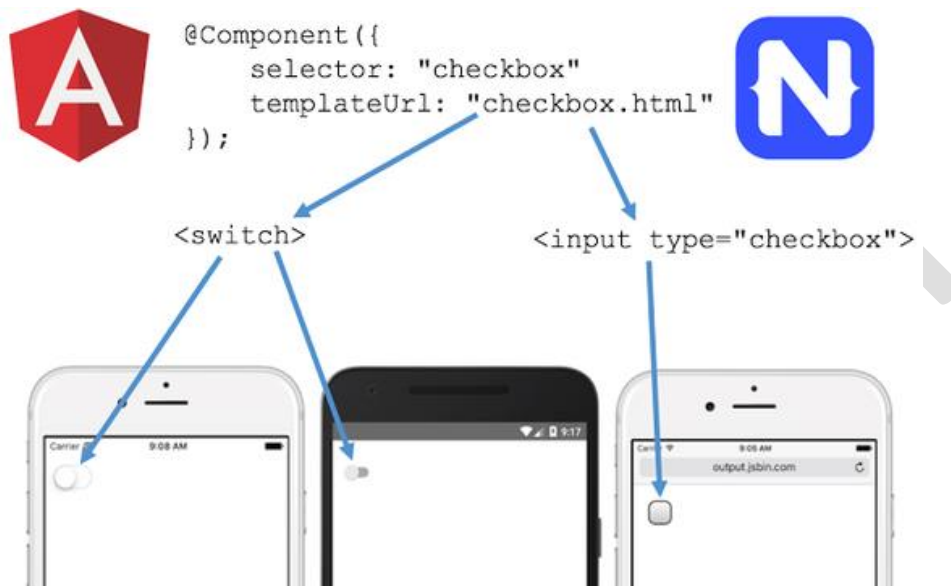
```
<div class="example2" ng-controller="ExampleController">  
  Name: <input ng-model="name" type="text"/>  
  <button ng-click="greet()">Greet</button>  
  <button ng-click="window.alert('Should not see me')">Won't greet</button>  
</div>
```



Comparison of AngularJS with Other Frameworks:

1) Angular 2 is mobile oriented & better in performance:

Angular 1.x was not built with mobile support in mind, where Angular 2 is mobile oriented. Angular 2 is using Hierarchical Dependency Injection system which is major performance booster. Angular 2 implements unidirectional tree based change detection which again increases performance. As per ng-conf meet up, angular 2 is 5 times faster as compared to angular 1.

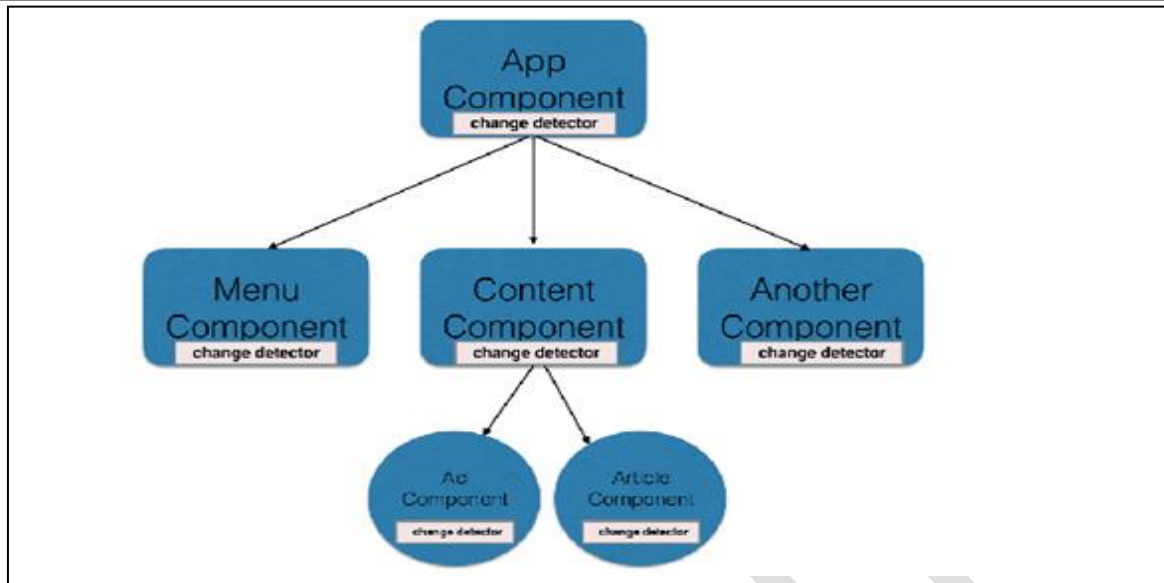


2) Angular 2 provides more choice for languages:

Angular 2 provides more choice for languages. You can use any of the languages from ES5, ES6, Typescript or Dart to write Angular 2 code. Where, Angular 1.x has ES5, ES6, and Dart. Using of Typescript is a great step as Typescript is awesome way to write JavaScript.

3) Angular 2 implements web standards like components:

Angular 2 implements web standards like components, and it provides better performance than Angular 1.



4) AngularJS 2.0 is not easy to setup as AngularJS 1.x:

AngularJS 1.x is easy to setup. All you need to do is to add reference of the library and you are good to go. Where AngularJS 2 is dependent of other libraries and it requires some efforts to set up it.

5) Angular 1.x controllers and \$scope are gone:

Angular 1.x controllers and \$scope are gone. We can say that controllers are replaced with "Components" in Angular 2. Angular 2 is component based. Angular 2 is using zone.js to detect changes.

Ex.

AngularJS.

```
var myApp = angular.module("myModule", []).controller("productController",
function($scope) {
    var prods = { name: "Prod1", quantity: 1 };
    $scope.products = prods;
});
```

Angular 2.

```
import { Component } from '@angular/core';
@Component({
    selector: 'prodsdata',
    template: '<h3>{{prods.name}}</h3>'
})
export class ProductComponent {
    prods = {name: 'Prod1', quantity: 1 };
}
```

Environmental Setup:

1. GitHub



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.

Ex:

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
<!doctype html>
<html ng-app="myapp" ng-controller="HelloController">

    <h2>Welcome {{helloTo.title}}!</h2>
<script>
    angular.module("myapp", [])
    .controller("HelloController", function($scope) {
        $scope.helloTo = {};
        $scope.helloTo.title = "Naresh IT- AngularJS";
    });
</script>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
</html>
```

Bower:

Keeping track of all these packages and making sure they are up to date (or set to the specific versions you need) is tricky. Bower to the rescue!

Bower can manage components that contain HTML, CSS, JavaScript, fonts or even image files. Bower doesn't concatenate or minify code or do anything else - it just installs the right versions of the packages you need and their dependencies.

To [get started](#), Bower works by fetching and installing [packages](#) from all over, taking care of hunting, finding, downloading, and saving the stuff you're looking for. Bower keeps track of these packages in a manifest file, [bower.json](#). How you use [packages](#) is up to you. Bower provides hooks to facilitate using packages in your [tools and workflows](#).

Bower is optimized for the front-end. If multiple packages depend on a package - jQuery for example - Bower will download jQuery just once. This is known as a flat dependency graph and it helps reduce page load.

Install the Bower.

```
$ npm install -g bower
```

Steps to Design the Static Angular Application:

Step 1: Load framework

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/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

```
<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/1.5.8/angular.min.js">
    </script>
  </body>
</html>
```

1. The *ng-app* directive indicates the start of AngularJS application.
2. 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.
3. The *ng-bind* then uses the *name* model to be displayed in the HTML ** tag whenever user enters input in the text box.
4. Closing *</div>* tag indicates the end of AngularJS application.

Directives:

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

1. ng-app:

- only one AngularJS application can be auto-bootstrapped per HTML document. The first ngApp found in the document will be used to define the root element to auto-bootstrap as an application. To run multiple applications in an HTML document you must manually bootstrap them using `angular.bootstrap` instead.
- AngularJS applications cannot be nested within each other.
- Do not use a directive that uses [transclusion](#) on the same element as ngApp. This includes directives such as ngIf, ngInclude and ngView. Doing this misplaces the app \$rootElement and the app's [injector](#), causing animations to stop working and making the injector inaccessible from outside the app.

2. ng-bind:

The ngBind attribute tells Angular to replace the text content of the specified HTML element with the value of a given expression, and to update the text content when the value of that expression changes.

Typically, you don't use ngBind directly, but instead you use the double curly markup like `{{expression}}` which is similar but less verbose.

It is preferable to use ngBind instead of `{{expression}}` if a template is momentarily displayed by the browser in its raw state before Angular compiles it. Since ngBind is an element attribute, it makes the bindings invisible to the user while the page is loading.

An alternative solution to this problem would be using the [ngCloak](#) directive.

3. ng-model:

The ngModel directive binds an input, select, text area (or custom form control) to a property on the scope using [NgModelController](#), which is created and exposed by this directive.

ngModel is responsible for:

- Binding the view into the model, which other directives such as input, text area or select require.
- Providing validation behavior (i.e. required, number, email, url).
- Keeping the state of the control (valid/invalid, dirty/pristine, touched/untouched, validation errors).
- Setting related css classes on the element (ng-valid, ng-invalid, ng-dirty, ng-pristine, ng-touched, ng-untouched, ng-empty, ng-not-empty) including animations.
- Registering the control with its parent [form](#).

4. ng-controller:

The ngController directive attaches a controller class to the view. This is a key aspect of how angular supports the principles behind the Model-View-Controller design pattern.

MVC components in angular:

- Model — Models are the properties of a scope; scopes are attached to the DOM where scope properties are accessed through bindings.
- View — The template (HTML with data bindings) that is rendered into the View.
- Controller — The ngController directive specifies a Controller class; the class contains business logic behind the application to decorate the scope with functions and values

5. ng-click:

The ngClick directive allows you to specify custom behavior when an element is clicked.

6. ng-dblclick:

The ngDbclick directive allows you to specify custom behavior on a dblclick event.

7. ng-submit:

Enables binding angular expressions to on submit events.

Additionally it prevents the default action (which for form means sending the request to the server and reloading the current page), but only if the form does not contain action, data-action, or x-action attributes.

8. ng-switch:

The ngSwitch directive is used to conditionally swap DOM structure on your template based on a scope expression. Elements within ngSwitch but without ngSwitchWhen or ngSwitchDefault directives will be preserved at the location as specified in the template.

9. ng-repeat:

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.

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.

Variable	Type	Details
\$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).

10. ng-options:

The ngOptions attribute can be used to dynamically generate a list of <option> elements for the <select> element using the array or object obtained by evaluating the ngOptions comprehension expression.

In many cases, [ngRepeat](#) can be used on <option> elements instead of ngOptions to achieve a similar result. However, ngOptions provides some benefits:

- more flexibility in how the <select>'s model is assigned via the select as part of the comprehension expression
- reduced memory consumption by not creating a new scope for each repeated instance
- increased render speed by creating the options in a document Fragment instead of individually

When an item in the <select> menu is selected, the array element or object property represented by the selected option will be bound to the model identified by the ngModel directive.

11. ng-mousedown:

The ngMousedown directive allows you to specify custom behavior on mouse down event.

12. ng-pattern:

ngPattern adds the pattern [validator](#) to [ngModel](#). It is most often used for text-based [input](#) controls, but can also be applied to custom text-based controls.

The validator sets the pattern error key if the [ngModel.\\$viewValue](#) does not match a RegExp which is obtained by evaluating the Angular expression given in the ngPattern attribute value:

- If the expression evaluates to a RegExp object, then this is used directly.
- If the expression evaluates to a string, then it will be converted to a RegExp after wrapping it in ^ and \$ characters. For instance, "abc" will be converted to new RegExp ('^abc\$').

13. ng-pluralize:

ngPluralize is a directive that displays messages according to en-US localization rules. These rules are bundled with angular.js, but can be overridden (see [Angular i18n](#) dev guide).

You configure ngPluralize directive by specifying the mappings between [plural categories](#) and the strings to be displayed.

14. ng-include:

By default, the template URL is restricted to the same domain and protocol as the application document. This is done by calling [\\$sce.getTrustedResourceUrl](#) on it. To load templates from other domains or protocols you may either [whitelist them](#) or [wrap them](#) as trusted values. Refer to Angular's [Strict Contextual Escaping](#).

In addition, the browser's [Same Origin Policy](#) and [Cross-Origin Resource Sharing \(CORS\)](#) policy may further restrict whether the template is successfully loaded. For example, ngInclude won't work for cross-domain requests on all browsers and for file:// access on some browsers.

15. ng-hide:

The ngHide directive shows or hides the given HTML element based on the expression provided to the ngHide attribute. The element is shown or hidden by removing or adding the ng-hide CSS class onto the element. The .ng-hide CSS class is predefined in AngularJS and sets the display style to none (using an !important flag). For CSP mode please adds angular-csp.css to your html file (see [ngCsp](#)).

16. ng-if:

The ngIf directive removes or recreates a portion of the DOM tree based on an {expression}. If the expression assigned to ngIf evaluates to a false value then the element is removed from the DOM, otherwise a clone of the element is reinserted into the DOM.

ngIf differs from ngShow and ngHide in that ngIf completely removes and recreates the element in the DOM rather than changing its visibility via the display css property. A common case when this difference is significant is when using css selectors that rely on an element's position within the DOM, such as the :first-child or :last-child pseudo-classes.

Note that when an element is removed using ngIf its scope is destroyed and a new scope is created when the element is restored. The scope created within ngIf inherits from its parent scope using [prototypal inheritance](#). An important implication of this is if ngModel is used within ngIf to bind to a JavaScript primitive defined in the parent scope. In this case any modifications made to the variable within the child scope will override (hide) the value in the parent scope.

Also, ngIf recreates elements using their compiled state. An example of this behavior is if an element's class attribute is directly modified after it's compiled, using something like

jQuery's .addClass () method, and the element is later removed. When ngIf recreates the element the added class will be lost because the original compiled state is used to regenerate the element.

Additionally, you can provide animations via the ngAnimate module to animate the enter and leave effects.

17. ng-init:

The ngInit directive allows you to evaluate an expression in the current scope.

18. ng-show:

The ngShow directive shows or hides the given HTML element based on the expression provided to the ngShow attribute. The element is shown or hidden by removing

or adding the `.ng-hide` CSS class onto the element. The `.ng-hide` CSS class is predefined in AngularJS and sets the display style to none (using an `important` flag). For CSP mode please add `angular-csp.css` to your html file (see [ngCsp](#)).

19. ngCloak:

The `ngCloak` directive is used to prevent the Angular html template from being briefly displayed by the browser in its raw (uncompiled) form while your application is loading. Use this directive to avoid the undesirable flicker effect caused by the html template display.

20. Ng-classeven:

The `ngClassOdd` and `ngClassEven` directives work exactly as [ngClass](#), except they work in conjunction with `ngRepeat` and take effect only on odd (even) rows.

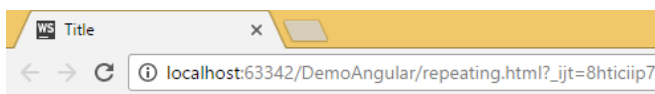
Examples Using Directives

ng-repeat

```
<!DOCTYPE html>
<html lang="en" ng-app="myModule">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script>

    var myApp=angular.module('myModule',[]);
    myApp.controller('myController', function ($scope) {
      var products= [
        {ProductId:1, Name:"TV", Price:3400},
        {ProductId:2, Name:"mobile", Price:12000},
      ];
      $scope.products=products;
    });
  </script>
</head>
<body ng-controller="myController">
  <table>
    <table border="1">
      <caption> Products List </caption>
      <thead>
        <tr>
          <td> Product ID </td>
          <td> Product Name </td>
          <td> Product Price </td>
```

```
</tr>
</thead>
<tbody>
<tr ng-repeat="product in products">
  <td>{{product.ProductId}}</td>
  <td>{{product.Name}}</td>
  <td>{{product.Price}}</td>
</tr>
</tbody>
</table>
</table>
</body>
</html>
```



Products List

Product ID	Product Name	Product Price
1	TV	3400
2	mobile	12000

Nested Repeat :

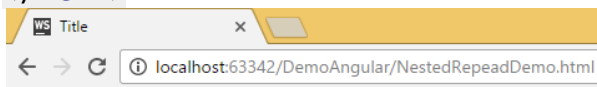
```
<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script>
    var myapp = angular.module('MyApp',[]) ;
    myapp.controller('MyController', function ($scope) {
      var collection=[
        {name:"Electronics",
          list:[{name:"Samsung TV"},{name:"Mobile"}]
        },
        {
          name:"Shoes",
          list:[{name:"Nike Shoe"},{name:"Lee Cooper"}]
        }
      ];
      $scope.collection = collection;
```



```

    })
  </script>
</head>
<body ng-controller="MyController">
  <ol>
    <li ng-repeat="category in collection" >
      {{category.name}}
      <ul>
        <li ng-repeat="item in category.list">
          {{item.name}}
        </li>
      </ul>
    </li>
  </ol>
</body>
</html>

```



1. Electronics
 - Samsung TV
 - Mobile
2. Shoes
 - Nike Shoe
 - Lee Cooper










Repeat with Event (Likes and Dislikes counter)

```

<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script>
    var myapp = angular.module('MyApp',[]) ;
    myapp.controller('MyController', function ($scope) {
      var pics = [
        {name:"Range Rower",image:"Images/3.jpg",
dislikes:0, likes:0},
        {name:"Ferrari",image:"Images/9.jpg", dislikes:0,
likes:0},
        {name:"Audi",image:"Images/1.jpg", dislikes:0,
likes:0},
      ];
    });
  </script>

```

```
        $scope.pics=pics;
        $scope.DislikesCounter = function (item) {
            item.dislikes++;
        }
        $scope.likesCounter = function (item) {
            item.likes++;
        }
    });
</script>
</head>
<body ng-controller="MyController">
    <table border="1" width="600">
        <thead>
            <tr>
                <td> Car Name </td>
                <td> Preview </td>
                <td> Dislikes </td>
                <td> Likes </td>
                <td> Dislike / Like </td>
            </tr>
        </thead>
        <tbody>
            <tr ng-repeat="item in pics">
                <td>{{item.name}}</td>
                <td>
            </td>
                <td>{{item.dislikes}}</td>
                <td>{{item.likes}}</td>
                <td>
                    <a href=""> </a>
                    
                </td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

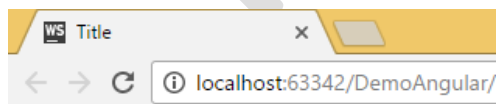
Car Name	Preview	Dislikes	Likes	Dislike / Like
Range Rower		1	3	 Dislike  Like
Ferrari		2	3	 Dislike  Like
Audi		2	3	 Dislike  Like

Ng-Checked Demo

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
</head>
<body ng-app="">
<p>Products :</p>
<input type="checkbox" ng-model="all"> Check all<br><br>
<input type="checkbox" ng-checked="all">TV<br>
<input type="checkbox" ng-checked="all">Mobile<br>
<input type="checkbox" ng-checked="all">Shoe
</body>
</html>

```



Products :

☒ Check all

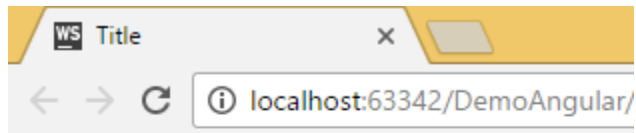
☒ TV

☒ Mobile

☒ Shoe

Ng-Class

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <style>
    .strike {
      text-decoration: line-through;
    }
    .bold {
      font-weight: bold;
    }
    .red {
      color: red;
    }
    .has-error {
      color: red;
      background-color: yellow;
    }
    .orange {
      color: orange;
    }
  </style>
</head>
<body ng-app="">
<p ng-class="{strike: deleted, bold: important, 'has-error':
error}">Select CheckBox to Apply</p>
<label>
  <input type="checkbox" ng-model="deleted">
  deleted (apply "strike" class)
</label><br>
<label>
  <input type="checkbox" ng-model="important">
  important (apply "bold" class)
</label><br>
<label>
  <input type="checkbox" ng-model="error">
  error (apply "has-error" class)
</label>
</body>
</html>
```



Select ~~CheckBox~~ to Apply

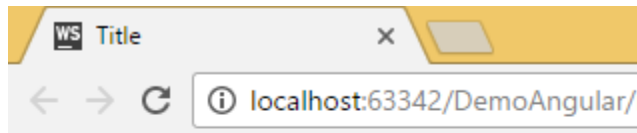
- ☒ deleted (apply "strike" class)
- ☒ important (apply "bold" class)
- ☒ error (apply "has-error" class)

Ng-Class Even / Odd

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <style>

    .odd {
      color: red;
    }
    .even {
      color: blue;
    }

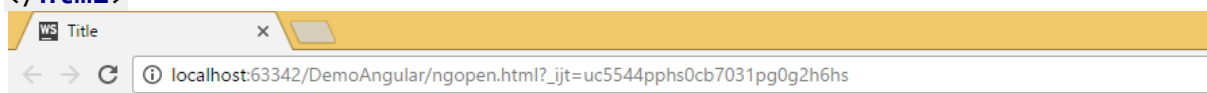
  </style>
</head>
<body ng-app="">
<ol ng-init="names=['John', 'Mary', 'David', 'Raj']">
  <li ng-repeat="name in names">
    <span ng-class-odd="'odd'" ng-class-even="'even'">
      {{name}} &nbsp; &nbsp; &nbsp;
    </span>
  </li>
</ol>
</body>
</html>
```



1. John
2. Mary
3. David
4. Raj

Ng-Open

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
</head>
<body ng-app="">
  <div>
    <input type="checkbox" ng-model="display"> Show / Hide Ads
  </div>
  <dialog ng-open="display">
    
    <br>
    <address>
      Pepsifoods ltd | Mumbai | 060-4959959
    </address>
  </dialog>
</body>
</html>
```



☒ Show / Hide Ads



Ng-Open Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
</head>
<body ng-app="">
  <h2>Table of Contents </h2>
  Angular Js <input type="checkbox" ng-model="angular"> <br>
  Jquery <input type="checkbox" ng-model="jquery"> <br>
  <br>
  <div>
    <details ng-open="angular">
      <summary>Angular Js</summary>
      <p>Angular is a framework what uses MVC </p>
    </details>
  </div>
  <div>
    <details ng-open="jquery">
      <summary>Jquery</summary>
      <p>Jquery is a library </p>
    </details>
  </div>
</body>
</html>
```

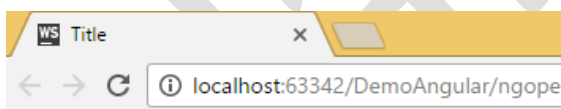


Table of Contents

Angular Js ☒

Jquery ☐

▼ Angular Js

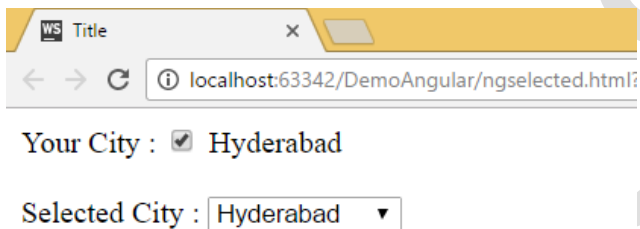
Angular is a framework what uses MVC

► Jquery

Ng-Selected

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
</head>
<body ng-app="">
  Your City : <input type="checkbox" ng-model="Selected">
  Hyderabad <br>

  <br>
  Selected City :
    <select>
      <option>Selected City</option>
      <option>Delhi</option>
      <option ng-selected="Selected">Hyderabad</option>
    </select>
</body>
</html>
```

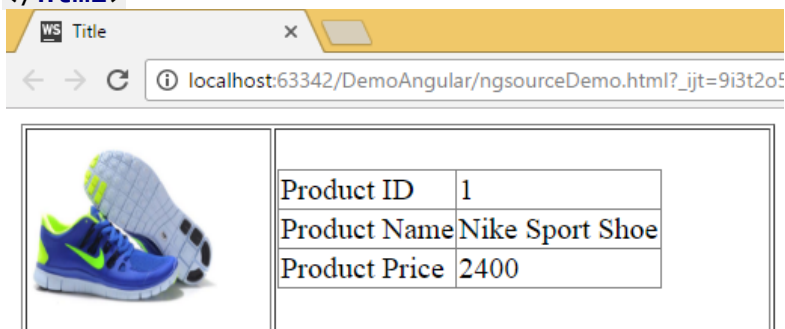


Ng-Source

```
<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script>
    var myapp = angular.module('MyApp', []) ;
    myapp.controller('MyController', function ($scope) {
      var product = {ProductId:1, Name:"Nike Sport Shoe",
Price:2400, Photo:"Images/shoe.jpg"};
      $scope.prod = product;
    })
  </script>
```

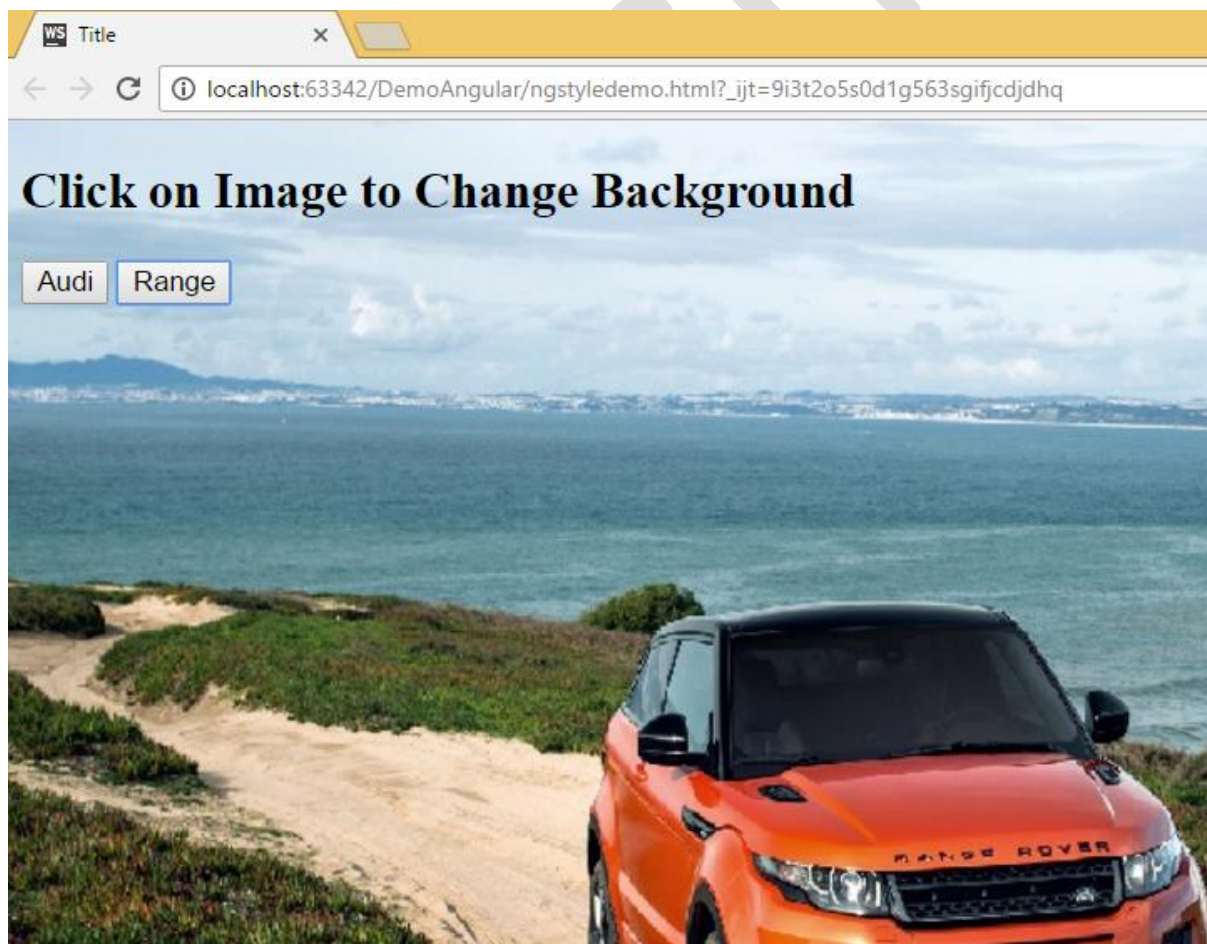


```
</script>
</head>
<body ng-controller="MyController">
  <table border="1" width="400">
    <tr>
      <td>
    </td>
    <td>
      <table rules="all" frame="box">
        <tr>
          <td>Product ID </td>
          <td>{{prod.ProductId}}</td>
        </tr>
        <tr>
          <td>Product Name </td>
          <td>{{prod.Name}}</td>
        </tr>
        <tr>
          <td>Product Price </td>
          <td>{{prod.Price}}</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
</body>
</html>
```



Ng-Style Demo

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
</head>
<body ng-app="" ng-style="mystyle">
  <h2>Click on Image to Change Background </h2>
  <button ng-click="mystyle={'background-
image':'url(images/1.jpg)'}">Audi</button>
  <button ng-click="mystyle={'background-
image':'url(images/3.jpg)'}">Range</button>
  <br> <br>
</body>
</html>
```



Ng-Include

```
<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script>
    var myapp = angular.module('MyApp',[]) ;
    myapp.controller('MyController', function ($scope) {
      var products= [
        {ProductId:1, Name:"Samsung TV", Price:30065.67,
Image:"Images/tv.jpg"},
        {ProductId:2, Name:"Nike Shoe", Price:6009.33,
Image:"Images/shoe.jpg"}
      ];
      $scope.products=products;
      $scope.view="catalogview.html";
    });
  </script>
</head>
<body ng-controller="MyController">
  Select a View :
  <select ng-model="view">
    <option value="gridview.html">Grid View</option>
    <option value="catalogview.html">Catalog View</option>
  </select>
  <div ng-include="view">

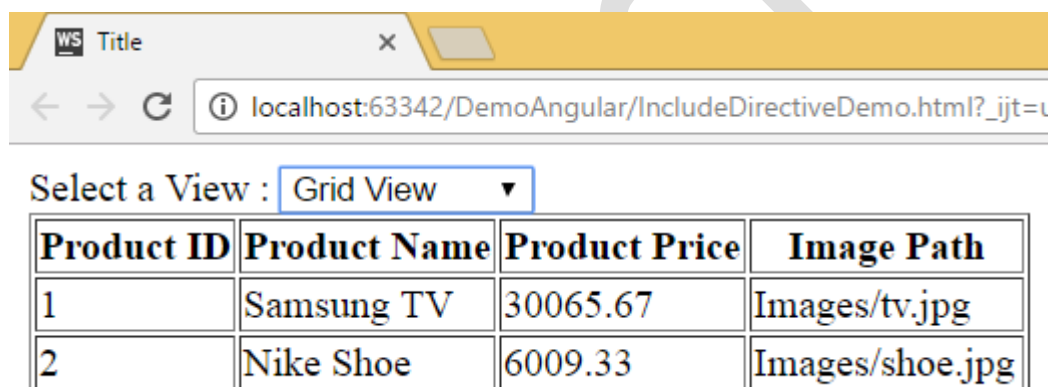
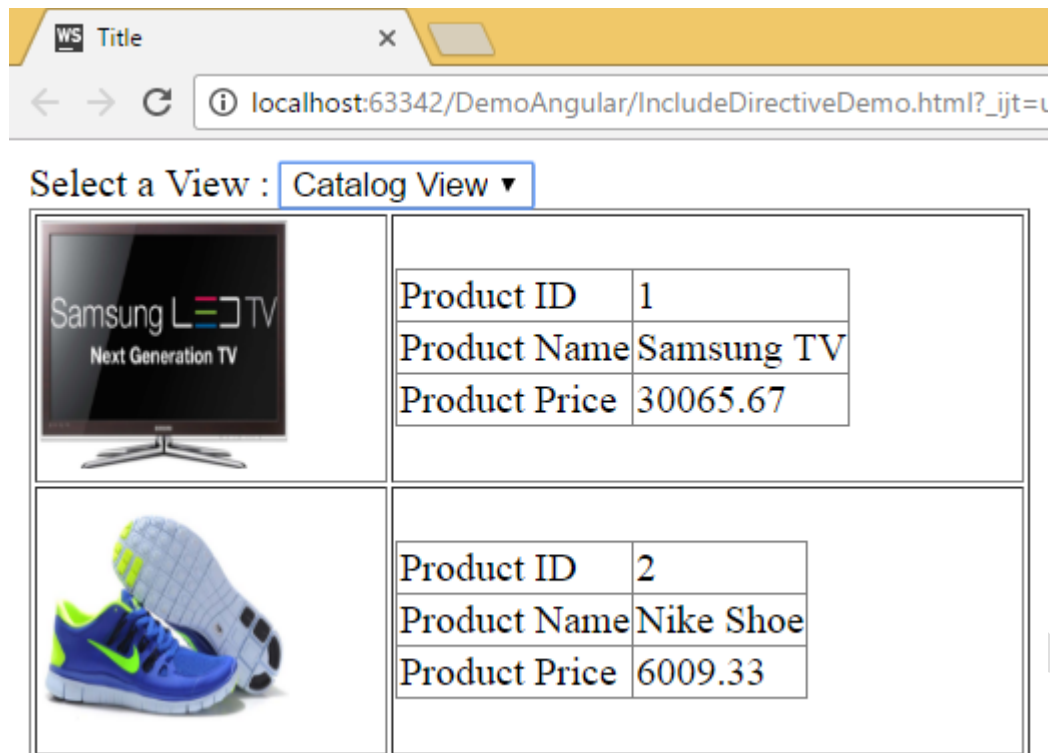
</div>
</body>
</html>
```

GridView Page

```
<table border="1" width="400">
  <th>Product ID </th>
  <th>Product Name </th>
  <th>Product Price </th>
  <th>Image Path </th>
  <tr ng-repeat="product in products">
    <td>{{product.ProductId}}</td>
    <td>{{product.Name}}</td>
    <td>{{product.Price}}</td>
    <td>{{product.Image}}</td>
  </tr>
</table>
```

Catalog view Page

```
<table border="1" width="400">
  <tr ng-repeat="product in products">
    <td> </td>
    <td>
      <table rules="all" frame="box">
        <tr>
          <td>Product ID </td>
          <td>{{product.ProductId}}</td>
        </tr>
        <tr>
          <td>Product Name </td>
          <td>{{product.Name}}</td>
        </tr>
        <tr>
          <td>Product Price </td>
          <td>{{product.Price}}</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```



Ng-Mouse Events

```
<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.min.js"></script>
  <script>
    var myapp=angular.module('MyApp',[]);
    myapp.controller('MyController', function ($scope) {
      $scope.pic="Images/Pepsi.jpg";
    });
  </script>
</head>
<body>
  <div>
    <img alt="Pepsi Logo" data-bbox="145 645 298 735"/>
  </div>
</body>
</html>
```

```

        $scope.Ad1 = function () {
            $scope.pic="Images/rDigital.jpg";
        }
        $scope.Ad2 = function () {
            $scope.pic="Images/Pepsi.jpg";
        }
    });
</script>
</head>
<body ng-controller="MyController">
    <div>
        <caption>Mouse the mouse over to change
        Advertisement</caption>
        
    </div>
</body>
</html>

```



Ng-Show

```

<!DOCTYPE html>
<html lang="en" ng-app="ProdModule">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script type="text/javascript" src="angular.js"></script>
    <script>
        var myapp=angular.module('ProdModule',[]);
        myapp.controller('ProdController', function ($scope) {
            $scope.txtId='';
            $scope.txtName='';
            $scope.txtPrice='';
            $scope.details='';
            $scope.IsVisible=false;
            $scope.ShowDetails= function () {
                $scope.IsVisible=true;
                var product={ProductId:$scope.txtId,
                _ProductName:$scope.txtName, ProductPrice:$scope.txtPrice};
            }
        });
    </script>

```

```

        $scope.prod=product;
    });
</script>
</head>
<body ng-controller="ProdController">
    <form>
        Product ID : <input type="text" ng-model="txtId">
        <br>
        Product Name : <input type="text" ng-model="txtName">
        <br>
        Product Price : <input type="text" ng-model="txtPrice">
        <br>
        <button ng-click="ShowDetails()">Show Details</button>
        <div ng-show="IsVisible">
            Product ID : {{prod.ProductId}} <br>
            Product Name : {{prod.ProductName}} <br>
            Product Price : {{prod.ProductPrice}}
        </div>
    </form>
</body>
</html>

```

Product ID : 1

Product Name : TV

Product Price : 45000

Show Details

Product ID : 1

Product Name : TV

Product Price : 45000

Ng-required

```

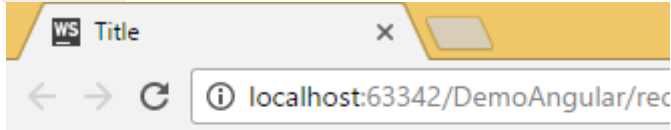
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script type="text/javascript" src="angular.js"></script>
</head>
<body ng-app="">
    <form name="form">
        <input type="checkbox" ng-model="required" id="required">
        Name required <br>
        Enter Name : <input type="text" name="input" id="input"

```

```

ng-model="txtname" ng-required="required">
  <div>
    Required : {{form.input.$error.required}} <br>
    Name : {{txtname}}
  </div>
</form>
</body>
</html>

```



☒ Name required

Enter Name :

Required : true

Name :

Ng-Key Events

```

<div>
  <input type="text" ng-keypress="event=$event">
  <br>
  Key Code : {{event.keyCode}} <br>
  Char Code : {{event.charCode}}
</div>

```

Ng-href

```

<div>
  <a ng-href="{{'/DemoAngular/ani.html'}}">Click Here</a>
</div>

```

Ng-Pattern

```

<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script>
    var myapp = angular.module('MyApp',[]) ;
    myapp.controller('MyController', function ($scope) {

```



```

        $scope.regex='\\d';
    })
</script>
</head>
<body ng-controller="MyController">
    <form name="form">
        <div>
            <input type="checkbox" ng-model="showDetails"><br>
            <br>

            <dialog ng-open="showDetails">
                <summary>Copyright 1999-2016.</summary>
                <p> - by Refsnes Data. All Rights Reserved.</p>
            </dialog>
        </div>
        <br>
        <div>
            Enter Elements : <input type="text" ng-model="products"
ng-list>
                {{products}}
            </div>
            <div>
                Name : <input type="text" name="txtname" ng-
model="txtname" ng-maxlength="5" ng-minlength="2">
                <span ng-if="!form.txtname.$valid">Name too short or
long.. max 5 chars only</span>
            </div>
            <div>
                Enter Your Pattern:
                <input type="text" id="regex" ng-model="regex">
            </div>
            <br>
            <div>
                Pattern Applied to TextBox :
                <input type="text" ng-model="model" id="input"
name="input" ng-pattern="regex">
            </div>
            <div>
                <h2>Pattern Status Is Valid : {{form.input.$valid}} </h2>
                <h2>Your Value {{ model }}</h2>
            </div>
        </form>
    </body>
</html>

```

Enter Your Pattern:

Pattern Applied to TextBox :

Pattern Status Is Valid : true

Your Value

Ng-Bind-Html

```
<!DOCTYPE html>
<html lang="en" ng-app="MyApp">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script type="text/javascript" src="angular.js"></script>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-
sanitize.js"></script>
  <script>
    var myapp = angular.module('MyApp', ['ngSanitize']) ;
    myapp.controller('MyController', function ($scope) {
      $scope.hello = "Hello " + "<b>" + "Welome" + "</b>";
    })
  </script>
</head>
<body ng-controller="MyController">
  <div ng-bind-html="hello"></div>
</body>
</html>
```

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.

Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its *compile()* method of the custom directive then process the element using *link()* method of the custom directive based on the scope of the directive. AngularJS provides support to create custom directives for following type of elements.

Element directives - Directive activates when a matching element is encountered.

Attribute - - Directive activates when a matching attribute is encountered.

CSS - - Directive activates when a matching css style is encountered.

Comment - - Directive 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 above custom 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.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 are using name attribute's value 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/1.3.14/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>
```

The AngularJS \$scope functions \$watch(), \$digest() and \$apply() are some of the central functions in AngularJS. Understanding \$watch(), \$digest() and \$apply() is essential in order to understand AngularJS.

When you create a data binding from somewhere in your view to a variable on the \$scope object, AngularJS creates a "watch" internally. A watch means that AngularJS watches changes in the variable on the \$scope object. The framework is "watching" the variable. Watches are created using the \$scope.\$watch() function which I will cover later in this text.

At key points in your application AngularJS calls the \$scope.\$digest() function. This function iterates through all watches and checks if any of the watched variables have changed. If a watched variable has changed, a corresponding listener function is called. The listener function does whatever work it needs to do, for instance changing an HTML text to reflect the new value of the watched variable. Thus, the \$digest() function is what triggers the data binding to update.

Most of the time AngularJS will call the `$scope.$watch()` and `$scope.$digest()` functions for you, but in some situations you may have to call them yourself. Therefore it is really good to know how they work.

The `$scope.$apply()` function is used to execute some code, and then call `$scope.$digest()` after that, so all watches are checked and the corresponding watch listener functions are called. The `$apply()` function is useful when integrating AngularJS with other code.

I will get into more detail about the `$watch()`, `$digest()` and `$apply()` functions in the remainder of this text.

`$watch()`

The `$scope.watch()` function creates a watch of some variable. When you register a watch you pass two functions as parameters to the `$watch()` function:

- A value function
- A listener function
-

Here is an example:

```
$scope.$watch(function() {},  
              function() {}  
              );
```

The first function is the value function and the second function is the listener function.

The value function should return the value which is being watched. AngularJS can then check the value returned against the value the watch function returned the last time. That way AngularJS can determine if the value has changed. Here is an example:

```
$scope.$watch(function(scope) { return scope.data.myVar },  
              function() {}  
              );
```

This example value function returns the `$scope` variable `scope.data.myVar`. If the value of this variable changes, a different value will be returned, and AngularJS will call the listener function.

Notice how the value function takes the `scope` as parameter (without the `$` in the name). Via this parameter the value function can access the `$scope` and its variables. The value function can also watch global variables instead if you need that, but most often you will watch a `$scope` variable.

The listener function should do whatever it needs to do if the value has changed. Perhaps you need to change the content of another variable, or set the content of an HTML element or something. Here is an example:

```
$scope.$watch(function(scope) { return scope.data.myVar },
  function(newValue, oldValue) {
    document.getElementById("").innerHTML =
      "" + newValue + "";
  }
);
```

This example sets the inner HTML of an HTML element to the new value of the variable, embedded in the `b` element which makes the value bold. Of course you could have done this using the code `{{ data.myVar }}`, but this is just an example of what you can do inside the listener function.

\$digest()

The `$scope.$digest()` function iterates through all the watches in the `$scope` object, and its child `$scope` objects (if it has any). When `$digest()` iterates over the watches, it calls the value function for each watch. If the value returned by the value function is different than the value it returned the last time it was called, the listener function for that watch is called.

The `$digest()` function is called whenever AngularJS thinks it is necessary. For instance, after a button click handler has been executed, or after an AJAX call returns (after the `done()` / `fail()` callback function has been executed).

You may encounter some corner cases where AngularJS does not call the `$digest()` function for you. You will usually detect that by noticing that the data bindings do not update the displayed values. In that case, call `$scope.$digest()` and it should work. Or, you can perhaps use `$scope.$apply()` instead which I will explain in the next section.

\$apply()

The `$scope.$apply()` function takes a function as parameter which is executed, and after that `$scope.$digest()` is called internally. That makes it easier for you to make sure that all watches are checked, and thus all data bindings refreshed. Here is an `$apply()` example:

```
$scope.$apply(function() {
  $scope.data.myVar = "Another value";
});
```

The function passed to the `$apply()` function as parameter will change the value of `$scope.data.myVar`. When the function exits AngularJS will call the `$scope.$digest()` function so all watches are checked for changes in the watched values.

Example:

To illustrate how `$watch()`, `$digest()` and `$apply()` works, look at this example:

```
<div ng-controller="myController">
  {{data.time}}
<br/>
<button ng-click="updateTime()">update time - ng-click</button>
<button id="updateTimeButton" >update time</button>
</div>
<script>
  var module    = angular.module("myapp", []);
  var myController1 = module.controller("myController", function($scope) {
    $scope.data = { time : new Date() };
    $scope.updateTime = function() {
      $scope.data.time = new Date();
    }
    document.getElementById("updateTimeButton")
      .addEventListener('click', function() {
        console.log("update time clicked");
        $scope.data.time = new Date();
      });
  });
</script>
```

This example binds the `$scope.data.time` variable to an interpolation directive which merges the variable value into the HTML page. This binding creates a watch internally on the `$scope.data.time` variable.

The example also contains two buttons. The first button has an `ng-click` listener attached to it. When that button is clicked the `$scope.updateTime()` function is called, and after that AngularJS calls `$scope.$digest()` so that data bindings are updated.

The second button gets a standard JavaScript event listener attached to it from inside the controller function. When the second button is clicked that listener function is executed. As you can see, the listener functions for both buttons do almost the same, but when the second button's listener function is called, the data binding is not updated. That is because the `$scope.$digest()` is not called after the second button's event listener is executed. Thus, if you click the second button the time is updated in the `$scope.data.time` variable, but the new time is never displayed.

To fix that we can add a `$scope.$digest()` call to the last line of the button event listener, like this:


```
document.getElementById("updateTimeButton")
    .addEventListener('click', function() {
        console.log("update time clicked");
        $scope.data.time = new Date();
        $scope.$digest();
    });
```

Instead of calling `$digest()` inside the button listener function you could also have used the `$apply()` function like this:

```
document.getElementById("updateTimeButton")
    .addEventListener('click', function() {
        $scope.$apply(function() {
            console.log("update time clicked");
            $scope.data.time = new Date();
        });
    });
```

Notice how the `$scope.$apply()` function is called from inside the button event listener, and how the update of the `$scope.data.time` variable is performed inside the function passed as parameter to the `$apply()` function. When the `$apply()` function call finishes AngularJS calls `$digest()` internally, so all data bindings are updated.

JSON:

1. JSON Stands for Java Script Object Notation.
2. JSON is lightweight data-interchange format.
3. JSON is easy to read and write than XML.
4. JSON is language independent.
5. JSON supports array, object, string, number and values.
6. The JSON file must be save with .json extension.

Ex.

first.json

```
{"employees":[  
  {"name":"NareshIT1", "email":"test@gmail.com"},  
  {"name":" NareshIT2", "email":" test @gmail.com"},  
  {"name":" NareshIT3", "email":" test @gmail.com"}  
]}
```

Features of JSON:

1. Simplicity
2. Openness
3. Self Describing
4. Internationalization
5. Extensibility
6. Interoperability

JSON vs XML:

A list of differences between JSON and XML are given below.

No.	JSON	XML
1)	JSON stands for JavaScript Object Notation.	XML stands for eXtensible Markup Language.
2)	JSON is simple to read and write.	XML is less simple than JSON.
3)	JSON is easy to learn.	XML is less easy than JSON.
4)	JSON is data-oriented.	XML is document-oriented.

5)	JSON doesn't provide display capabilities.	XML provides the capability to display data because it is a markup language.
6)	JSON supports array.	XML doesn't support array.
7)	JSON is less secured than XML.	XML is more secured.
8)	JSON files are more human readable than XML.	XML files are less human readable.
9)	JSON supports only text and number data type.	XML support many data types such as text, number, images, charts, graphs etc. Moreover, XML offeres options for transferring the format or structure of the data with actual data.

JSON Example:

```
{"employees":[
  {"name":"Vimal", "email":"vjaiswal1987@gmail.com"},
  {"name":"Rahul", "email":"rahul12@gmail.com"},
  {"name":"Jai", "email":"jai87@gmail.com"}
]}
```

XML Example:

```
<employees>
  <employee>
    <name>Vimal</name>
    <email>vjaiswal1987@gmail.com</email>
  </employee>
  <employee>
    <name>Rahul</name>
    <email>rahul12@gmail.com</email>
  </employee>
  <employee>
    <name>Jai</name>
    <email>jai87@gmail.com</email>
  </employee>
</employees>
```

Similarities between JSON and XML

1. Both are simple and open.
2. Both supports unicode. So internationalization is supported by JSON and XML both.
3. Both represents self describing data.
4. Both are interoperable or language-independent.

JSON Object Example

```
{  
  "employee": {  
    "name": "sonoo",  
    "salary": 56000,  
    "married": true  
  }  
}
```

JSON Array example:

The [(square bracket) represents the JSON array. A JSON array can have values and objects.

Let's see the example of JSON array having values.

```
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

Let's see the example of JSON array having objects.

```
[  
  {"name": "Ram", "email": "Ram@gmail.com"},  
  {"name": "Bob", "email": "bob32@gmail.com"}  
]
```

Services:

In AngularJS you can make your own service, or use one of the many built-in services.

Q) What is a Service?

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services. One of them is the \$location service.

The \$location service has methods which return information about the location of the current web page

Q) Why use Services?

For many services, like the \$location service, it seems like you could use objects that are already in the DOM, like the window.location object, and you could, but it would have some limitations, at least for your AngularJS application.

AngularJS constantly supervises your application, and for it to handle changes and events properly, AngularJS prefers that you use the \$location service instead of the window.location object.

HTML does not support embedding html pages within html page. To achieve this functionality following ways are used:

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 embedd HTML pages within a 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>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<style>
table, th , td {
    border: 1px solid grey;
    border-collapse: collapse;
    padding: 5px;
}
table tr:nth-child(odd) {
    background-color: #f2f2f2;
}
table tr:nth-child(even) {
    background-color: #ffffff;
}
</style>
</head>
<body>
<h2>AngularJS Sample Application</h2>
<div ng-app="mainApp" ng-controller="studentController">
<div ng-include="main.htm"></div>
<div ng-include="subjects.htm"></div>
</div>
<script>
var mainApp = angular.module("mainApp", []);
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;
    }
  };
});
</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>
```

AngularJS provides \$http control which works as a service to read data from the server. The 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>
```

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

ng-view tag 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

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

\$routeProvider is the key service which set the configuration of urls, map them with the corresponding html page or ng-template, and attach a controller with the same.

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>
```

Usage

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'
    });
});
```

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

\$routeProvider is defined as a function under config of mainApp module using key as '\$routeProvider'.

\$routeProvider.when defines a url "/addStudent" which then is mapped to "addStudent.htm". addStudent.htm should be present in the same path as main html page. If htm page is not defined then ng-template to be used with id="addStudent.htm". We've used ng-template.

"otherwise" is used to set the default view.

"controller" is used to set the corresponding controller for the view.

Example

Following example will showcase all the above mentioned directives.

testAngularJS.htm

```
<html>
<head>
  <title>Angular JS Views</title>
  <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
```

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

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

```
<script>  
  var mainApp = angular.module("mainApp", []);  
  mainApp.controller("shapeController", function($scope) {  
    $scope.message = "In shape controller";  
    $scope.type = "Shape";  
  });  
</script>
```

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

\$scope is passed as first argument to controller during its constructor definition.

\$scope.message and \$scope.type are the models which are to be used in the HTML page.

We've set values to models which will be reflected in the application module whose controller is shapeController.

We can define functions as well in \$scope.

Scope Inheritance

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

```
<script>  
  var mainApp = angular.module("mainApp", []);  
  mainApp.controller("shapeController", function($scope) {  
    $scope.message = "In shape controller";  
    $scope.type = "Shape";  
  });  
  mainApp.controller("circleController", function($scope) {  
    $scope.message = "In circle controller";  
  });  
</script>
```

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

We've set values to models in shapeController.

We've overridden message in child controller circleController. When "message" is used within module of controller circleController, the overridden message will be used.

Example:

Following example will showcase all the above mentioned directives.

```
testAngularJS.htm
<html>
<head>
  <title>Angular JS Forms</title>
</head>
<body>
  <h2>AngularJS Sample Application</h2>
  <div ng-app="mainApp" ng-controller="shapeController">
    <p>{{message}} <br/> {{type}} </p>
    <div ng-controller="circleController">
      <p>{{message}} <br/> {{type}} </p>
    </div>
    <div ng-controller="squareController">
      <p>{{message}} <br/> {{type}} </p>
    </div>
  </div>
  <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
  <script>
    var mainApp = angular.module("mainApp", []);
    mainApp.controller("shapeController", function($scope) {
      $scope.message = "In shape controller";
      $scope.type = "Shape";
    });
    mainApp.controller("circleController", function($scope) {
      $scope.message = "In circle controller";
    });
    mainApp.controller("squareController", function($scope) {
      $scope.message = "In square controller";
      $scope.type = "Square";
    });
  </script>
</body>
</html>
```

AngularJS supports the concepts of "Seperation of Concerns" using services architecture. Services are javascript functions and are responsible to do a specific tasks only. This makes them an individual entity which is maintainable and testable. Controllers, filters can call them as on requirement basis. Services are normally injected using dependency injection mechanism of AngularJS.

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

There are two ways to create a service.

- Factory
- Service

Using factory Method:

Using factory 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:

Using service method, we define a service and then assign method to it. We've also injected an already available service to it.

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

Example:

Following example will showcase all the above mentioned directives.

```
testAngularJS.htm
<html>
<head>
  <title>Angular JS Services</title>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
</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>
    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>
```


The \$http Service:

The \$http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

The \$timeout Service:

The \$timeout service is AngularJS' version of the window.setTimeout function.

The \$interval Service

The \$interval service is AngularJS' version of the window.setInterval function.

\$q:

A service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing.

Create Your Own Service

To create your own service, connect your service to the module

```
app.service('nareshIT', function(){  
    this.myFunc= function (x){  
        return x.toString(16);  
    }  
});
```

Dependency Injection is a software design pattern 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 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.

7. value
8. factory
9. service
10. provider
11. constant

1. value:

value is simple javascript object and it is used to pass values to controller during config phase.

```
//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);  
    }  
});
```

2. factory:

factory is a function which is used to return value. It creates value on demand whenever a service or controller requires. It normally 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. Services are defined using service() functions and then injected into 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 config phase(phase during which AngularJS bootstraps itself). Below mention script can be used to create MathService that we've created earlier. Provider is a special factory method with a method get() 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 config phase considering the fact that value can not be used to be passed during config phase.

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

Example:

Following example will showcase 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/1.3.14/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>
```

AngularJS Form Validations:

Form Validation

AngularJS offers client-side form validation.

AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.

AngularJS also holds information about whether they have been touched, or modified, or not.

You can use standard HTML5 attributes to validate input, or you can make your own validation functions.

Form State and Input State

- `$untouched` - The field has not been touched yet
- `$touched` - The field has been touched
- `$pristine` - The field has not been modified yet
- `$dirty` - The field has been modified
- `$invalid` - The field content is not valid
- `$valid` - The field content is valid
- `$pristine` - No fields have been modified yet
- `$dirty` - One or more have been modified
- `$invalid` - The form content is not valid
- `$valid` - The form content is valid
- `$submitted` - The form is submitted

CSS Classes:

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

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

The following classes are added to, or removed from, forms:

- `ng-pristine` - No fields has not been modified yet
- `ng-dirty` - One or more fields has been modified
- `ng-valid` - The form content is valid

- ng-invalid - The form content is not valid
- ng-valid-key - One key for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
- ng-invalid-key Example: ng-invalid-required

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

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.

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:

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

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

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>
```

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>
<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", []); Angular JS Tutorial 84
mainApp.controller('StudentController', function($scope) {
  $scope.fees = 100;
  $scope.admissiondate = new Date();
  $scope.rollno = 123.45;
});
</script>
</body>
</html>
```

Programs:**Example_1****Bower.json**

```
-  
{  
  "name": "angular-seed",  
  "description": "A starter project for AngularJS",  
  "version": "0.0.0",  
  "homepage": "https://github.com/angular/angular-seed",  
  "license": "MIT",  
  "private": true,  
  "dependencies": {  
    "angular": "~1.5.0"  
  }  
}
```

Bowerc.json

```
-  
{  
  "directory": "bower_components"  
}
```

Index.html:

```
<!DOCTYPE html>  
<html lang="en" ng-app>  
  <b>{{'Welcome to AngularJS'}}</b>  
  <!--<script src="bower_components/angular/angular.min.js"></script>-->  
  <!-- <script src="lib/angular.min.js"></script>-->  
  <script  
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>  
</html>
```

Example_2:**Bower.json**

```
{  
  "name": "angular-seed",  
  "description": "A starter project for AngularJS",  
  "version": "0.0.0",  
  "homepage": "https://github.com/angular/angular-seed",  
  "license": "MIT",  
  "private": true,  
  "dependencies": {  
    "angular": "~1.5.0"  
  }  
}
```


Bowerrc.json:

```
{
  "directory": "bower_components"
}
```

Index.html:

```
<!DOCTYPE html>
<html lang="en" ng-app>
  <input type="number" ng-model="model_one"> <br><br>
  <input type="number" ng-model="model_two"> <br><br>
  <h1 ng-bind="model_one+model_two"></h1>
  <script src="bower_components/angular/angular.min.js"></script>
</html>
```

Example_3:

Bower.json

```
-
{
  "name": "angular-seed",
  "description": "A starter project for AngularJS",
  "version": "0.0.0",
  "homepage": "https://github.com/angular/angular-seed",
  "license": "MIT",
  "private": true,
  "dependencies": {
    "angular": "~1.5.0"
  }
}
```

Bowerrc.json:

```
-
{
  "directory": "bower_components"
}
```

Index.html:

```
-
<!DOCTYPE html>
<html lang="en" ng-app="app">
  <div ng-controller="ctrl">
    {{var_one}}
  </div>
  <script src="bower_components/angular/angular.min.js"></script>
  <script src="app.js"></script>
  <script src="controllers/ctrl.js"></script>
</html>
```

App.js:

```
-  
var app=angular.module('app',[]);  
  
ctrl.js:  
-  
app.controller('ctrl',ctrl);  
ctrl.$inject=['$scope'];  
function ctrl($scope) {  
    $scope.var_one='I am from Controller';  
};
```

Example_4:

Ng-app

ngAppDemo_1.html

```
<!DOCTYPE html>  
<html lang="en">  
    <div ng-app>  
        {{'This is inside ng-app directive'}}  
        <br><br>  
        {{'This is also Inside ng-app directive'}}  
    </div>  
    <br><br>  
    <div>  
        {{'This is outside og ng-app directive'}}  
    </div>  
    <script  
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>  
</html>
```

ngAppDemo_2.html:

```
<!DOCTYPE html>  
<html lang="en">  
    <div ng-app="app" ng-controller="ctrl">  
        <b>I can Add:</b> {{a}}+{{b}}={{a+b}}  
        <br><br>  
        <div>  
            <b>I Can Subtract:</b> {{b}}-{{a}}={{b-a}}  
        </div>  
    </div>  
    <br><br>  
    <div>  
        <b>I Can't Multiply:</b> {{a}}*{{b}}={{a*b}}  
    </div>  
    <script  
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
```

```
<script src="ngAppDemo_2_Module.js"></script>
<script src="ngAppDemo_2_Controller.js"></script>
</html>
```

ngAppDemo_2_Module.js:

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

ngAppDemo_2_Controller.js:

```
-
app.controller('ctrl',ctrl);
ctrl.$inject=['$scope'];
function ctrl($scope) {
    $scope.a=10;
    $scope.b=20;
}
```

ngAppDemo_3.html:

```
<!DOCTYPE html>
<html lang="en">
    <div ng-controller="ctrl">
        <b>I can Add:</b>{{a}}+{{b}}={{a+b}}
    </div>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
    <script src="ngAppDemo_3_Module.js"></script>
    <script src="ngAppDemo_3_Controller.js"></script>
    <script src="ngAppDemo_3_Manual_Bootstrap.js"></script>
</html>
```

ngApp_Demo_3_Module.js:

```
var app=angular.module('app',[]);
ngAppDemo_3_Controller.js:
app.controller('ctrl',ctrl);
ctrl.$inject=['$scope'];
function ctrl($scope) {
    $scope.a=10;
    $scope.b=20;
}
```

ngAppDemo_3_Manual_Bootstrap.js:

```
angular.element(document).ready(function () {
    angular.bootstrap(document,['app']);
});
```

```
});
```

ngAppDemo_4.html:

-

```
<!DOCTYPE html>
<html lang="en">
  <div ng-app="app" ng-controller="ctrl">
    <b>I Can Add. </b>{{a}}+{{b}}={{a+b}}
  </div>
  <br><br>
  <div id="mydiv" ng-controller="myctrl">
    <b>I Can Subtract. </b>{{b}}-{{a}}={{b-a}}
  </div>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
    <script src="ngAppDemo_4_Module.js"></script>
    <script src="ngAppDemo_4_Controller.js"></script>
    <script src="ngAppDemo_4_ManualBootstrap.js"></script>
  </html>
```

ngAppDemo_4_Module.js:

```
var app=angular.module('app',[]);
ngAppDemo_4_Controller.js:
app.controller('ctrl',ctrl);
ctrl.$inject=['$scope'];
function ctrl($scope) {
  $scope.a=10;
  $scope.b=20;
}
app.controller('myctrl',myctrl);
myctrl.$inject=['$scope'];
function myctrl($scope) {
  $scope.a=10;
  $scope.b=20;
}
```

ngAppDemo_4_Manual_Bootstrap.js:

```
var mydiv=document.getElementById("mydiv");
angular.element(mydiv).ready(function () {
  angular.bootstrap(mydiv,['app']);
});
```

Filters:**Bower.json:**

```
{
  "name": "angular-seed",
  "description": "A starter project for AngularJS",
  "version": "0.0.0",
  "homepage": "https://github.com/angular/angular-seed",
  "license": "MIT",
  "private": true,
  "dependencies": {
    "angular": "~1.5.0",
    "bootstrap": "~3.3.7"
  }
}
```

Bowerrc.json:

```
{
  "directory": "bower_components"
}
```

Index.html:

```
<!DOCTYPE html>
<html lang="en" ng-app="app" ng-controller="ctrl">
  <input type="text" ng-model="my_model"> <br><br>
  <table class="table table-bordered">
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Age</th>
    </tr>
    <tr ng-repeat="x in data | orderBy:'-id' | limitTo:3 | filter:my_model">
      <td>{{x.id}}</td>
      <td>{{x.name}}</td>
      <td>{{x.age}}</td>
    </tr>
  </table>
  <hr>
  {{todayDate}}<br>
  <hr>
  <br><br>
  {{var_one | uppercase}}
  <hr>
  {{var_two | lowercase}}
  <b style="color: red">{{todayDate | date:'short'}}</b><br>
  <b style="color: green">{{todayDate | date:'medium'}}</b><br>
```

```
<b style="color: #0f0f0f">{{todayDate | date:'fullDate'}}</b><br>
<b style="color: #0f0f0f">{{todayDate | date:'dd-MM-yyyy'}}</b><br>
<b style="color: #0f0f0f">{{todayDate | date:'dd/MM/yy'}}</b><br>
<br><br>
{{jsonData | json}}
<br><br>
{{amount | currency:'$':5}}
<link rel="stylesheet"
href="bower_components/bootstrap/dist/css/bootstrap.min.css">
<script src="bower_components/angular/angular.min.js"></script>
<script src="app.js"></script>
<script src="controllers/ctrl.js"></script>
</html>
```

App.js:

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

ctrl.js:

```
app.controller('ctrl',ctrl);
ctrl.$inject=['$scope'];
function ctrl($scope) {
    $scope.data=[
        {id:1,name:'Hello_1',age:20},
        {id:3,name:'Hello_3',age:24},
        {id:2,name:'Hello_2',age:22},
        {id:5,name:'Hello_5',age:28},
        {id:4,name:'Hello_4',age:26}
    ];
    $scope.todayDate=new Date();
    $scope.var_one='hello';
    $scope.var_two='HELLO';
    $scope.jsonData={name:'Hello_1',id:1};
    $scope.amount=100;
}
```

Service_Example_One:

Bower.json"

```
{
  "name": "angular-seed",
  "description": "A starter project for AngularJS",
  "version": "0.0.0",
  "homepage": "https://github.com/angular/angular-seed",
  "license": "MIT",
  "private": true,
```

```
"dependencies": {  
  "angular": "~1.5.0"  
}
```

Bowerrc.json:

```
{  
  "directory": "bower_components"  
}
```

Index.html:

```
<!DOCTYPE html>  
<html lang="en" ng-app="app" ng-controller="ctrl">  
  <div class="container">  
    <table class="table table-bordered">  
      <tr>  
        <th>Name</th>  
        <th>City</th>  
        <th>Country</th>  
      </tr>  
      <tr ng-repeat="x in var_one">  
        <td>{{x.Name}}</td>  
        <td>{{x.City}}</td>  
        <td>{{x.Country}}</td>  
      </tr>  
    </table>  
  </div>  
  <button ng-click="getData()">Get Data</button>  
  <link rel="stylesheet" href="bootstrap.min.css">  
  <script src="bower_components/angular/angular.min.js"></script>  
  <script src="app.js"></script>  
  <script src="my_service.js"></script>  
  <script src="ctrl.js"></script>  
</html>
```

App.js:

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

ctrl.js:

```
app.controller('ctrl',ctrl);  
ctrl.$inject=['$scope','my_service'];
```

My_service.js:

Ui.router:

Index.html:

Naresh i Technologies, Opp. Satyam Theatre, Ameerpet, Hyd, Ph: 040-23746666, www.nareshit.com ::72::

App.js:

```
var app=angular.module('app',['ui.router']);
app.config(config);
config.$inject=['$stateProvider','$urlRouterProvider'];
function config($stateProvider,$urlRouterProvider) {
  $stateProvider
    .state("page_1",{
      url:'/page_1',
      templateUrl:'templates/page_1.html',
      controller:'page_1_controller'
    })
    .state("page_2",{
      url:'/page_2',
      templateUrl:'templates/page_2.html',
      controller:'page_2_controller'
    });
  $urlRouterProvider.otherwise('/page_1');
}
```

Page_1_controller.js

```
-
app.controller('page_1_controller',page_1_controller);
page_1_controller.$inject=['$scope'];
function page_1_controller($scope) {
  $scope.var_one='this is from controller_one';
};
```

Page_2_controller.js:

```
app.controller('page_2_controller',page_2_controller);
page_2_controller.$inject=['$scope'];
function page_2_controller($scope) {
  $scope.var_one='this is from controller_two';
};
```

Page_1.html:

```
<!DOCTYPE html>
<html lang="en">
  {{var_one}}
</html>
```

Page_2.html:

```
<!DOCTYPE html>
<html lang="en">
```

```
    {{var_one}}  
</html>
```

Communication Between Controllers:

Example_1:

Index.html

```
<!DOCTYPE html>  
<html lang="en" ng-app="app">  
  <div ng-controller="ctrl_one">  
    <button ng-click="clickMe('Hello')">Click Me</button>  
  </div>  
  <br><br>  
  <div ng-controller="ctrl_two">  
    {{msg}}  
  </div>  
  <script  
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>  
  <script src="appmodule.js"></script>  
  <script src="ctrl_one.js"></script>  
  <script src="ctrl_two.js"></script>  
</html>
```

App.js:

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

ctrl_one.js:

```
app.controller('ctrl_one',ctrl_one);  
ctrl_one.$inject=['$scope'];  
function ctrl_one($scope){  
  $scope.clickMe=function(data){  
    $scope.myfunction(data);  
  }  
}
```

Ctrl_two.js:

```
app.controller('ctrl_two',ctrl_two);  
ctrl_two.$inject=['$scope','$rootScope'];  
function ctrl_two($scope,$rootScope){  
  $rootScope.myfunction=function(data){  
    $scope.msg=data;  
  }  
}
```

```
}
```

Example_2:

Index.html:

```
<!DOCTYPE html>
<html lang="en" ng-app="app">
  <div ng-controller="parent_controller">
    <input type="button" ng-click="broadcast()" value="BroadCast">
    <br><br>
    <div ng-controller="child_one_controller">
      {{child_one}}
      <br><br>
      <button ng-click="childBroadCast()">Child BroadCast</button>
      <br><br>
      <div ng-controller="sub_child_one_controller">
        {{sub_child_one}}
      </div>
    </div>
    <br><br>
    <div ng-controller="child_two_controller">
      {{child_two}}
    </div>
  </div>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
  <script src="appmodule.js"></script>
  <script src="parent_controller.js"></script>
  <script src="child_one_controller.js"></script>
  <script src="child_two_controller.js"></script>
  <script src="sub_child_one_controller.js"></script>
</html>
```

App.js:

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

parent_controller.js:

```
app.controller('parent_controller',parent_controller);
parent_controller.$inject=['$scope'];
function parent_controller($scope) {
  $scope.broadcast = function () {
    var child_one={name:'Hello1'};
```

```
    var child_two={name:'Hello2'};
    $scope.$broadcast('key1',child_one);
    $scope.$broadcast('key2',child_two);
  };
}
```

Child_one_controller.js:

```
app.controller('child_one_controller',child_one_controller);
child_one_controller.$inject=['$scope'];
function child_one_controller($scope) {
  $scope.$on('key1',function(event,args){
    $scope.child_one=args.name;
  });
  $scope.childBroadCast = function(){
    $scope.$broadcast('key_one','This is from Child Broadcast');
  };
};
```

Child_two_controller.js:

```
app.controller('child_two_controller',child_two_controller);
child_two_controller.$inject=['$scope'];
function child_two_controller($scope) {
  $scope.$on('key2',function(event,args){
    $scope.child_two=args.name;
  });
};
```

Sub_child_one_controller.js:

```
app.controller('sub_child_one_controller',sub_child_one_controller);
sub_child_one_controller.$inject=['$scope'];
function sub_child_one_controller($scope) {
  $scope.$on('key_one',function (event,args) {
    $scope.sub_child_one=args;
  });
};
```

AngularJS With NodeJS Server:

In this Topic We Will Use AngularJS as a Front End and NodeJS as Server and Either MySQL or MongoDB As the database.

Example_1:

Lanuching the Angular Application By using NodeJS.

Directory Structure

Node_01

Index.html

Package.json

Server.js

Index.html

```
<!DOCTYPE html>
```

```
<html lang="en" ng-app>
```

```
  <h1 style="color: red">{{"Welcome to AngularJS - NodeJS"}}</h1>
```

```
  <script
```

```
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js"></script>
```

```
</html>
```

Package.json

-

```
{
```

```
  "name": "ExpressApp",
```

```
  "version": "0.0.1",
```

```
  "description": "A Node.js App using express",
```

```
  "dependencies": {
```

```
    "express": "^4.14.0"
```

```
  },
```

```
  "engine": "node >=0.6.x"
```

```
}
```

Server.js

```
var express=require("express");
```

```
var app=express();
```

```
app.use(express.static(__dirname+"/../Node_01"));
```

```
app.get("/",function (req,res) {
```

```
  res.redirect("/index.html");
```

```
});
```

```
app.listen(90);
```

```
console.log("Server Listening the Port No.90");
```

Example_2:

Reading the Data From file by using NodeJS.

Directory Structure

Node_02

list.json

package.json

server.json

list.json

```
{
  key1:'Hello_1',
  key2:'Hello_2',
  key3:'Hello_3'
}
```

Package.json

```
{
  "name": "ExpressApp",
  "version": "0.0.1",
  "description": "A Node.js App using express",
  "dependencies": {
    "express": "^4.14.0"
  },
  "engine": "node >=0.6.x"
}
```

Server.js

```
var express=require("express");
var fs=require("fs");
var app=express();
app.use(express.static(__dirname+"/../Node_02"));
app.get("/",function (req,res) {
  fs.readFile(__dirname+"/list.json",
    function (err,data) {
      console.log(data.toString());
      res.send(data.toString());
    });
});
app.listen(2000);
console.log("Server Listening the Port No.2000");
```

Example_3:

Reading the Data From json file.

Node_03

Index.html

app.js

services

my_service.js

controllers

ctrl.js

bower.json

.bowerrc.json

Package.json

emp.json

server.js

Index.html

<!DOCTYPE html>

<html lang="en" ng-app="app" ng-controller="ctrl">

<button ng-click="getEmp()">GetEmp</button>

<table border="1">

<tr>

<th>ID</th>

<th>Name</th>

<th>Age</th>

</tr>

<tr>

<td>{{info.id}}</td>

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

<td>{{info.age}}</td>

</tr>

</table>

<script src="bower_components/angular/angular.min.js"></script>

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

<script src="services/my_service.js"></script>

<script src="controllers/ctrl.js"></script>

</html>

app.js:

```
var app=angular.module("app",[]);
controllers/ctrl.js
app.controller("ctrl",ctrl);
ctrl.$inject=["$scope","my_service"];
function ctrl($scope,my_service) {
    $scope.getEmp=function () {
        my_service.my_fun().then(function (response) {
            $scope.info=response;
        });
    };
};
```

Services/my_service.js

```
app.service("my_service",my_service);
my_service.$inject=["$http"];
function my_service($http) {
    this.my_fun=function () {
        return $http.get("/employee").then(function (response) {
            return response.data;
        });
    };
};
```

bowerrc.json

```
{
  "directory": "bower_components"
}
```

Bower.json

```
{
  "name": "angular-seed",
  "description": "A starter project for AngularJS",
  "version": "0.0.0",
  "homepage": "https://github.com/angular/angular-seed",
  "license": "MIT",
  "private": true,
  "dependencies": {
    "angular": "~1.6.0"
  }
}
```


Emp.json

```
{
  "id":1,
  "name": "Hello_1",
  "age": 20
}
```

Package.json

```
{
  "name": "ExpressApp",
  "version": "0.0.1",
  "description": "A Node.js App using express",
  "dependencies": {
    "express": "^4.14.0"
  },
  "engine": "node >=0.6.x"
}
```

Server.js

```
var express=require("express");
var fs=require("fs");
var app=express();
app.use(express.static(__dirname+"/../Node_03"));
app.get("/",function (req,res) {
  res.redirect("/index.html");
});
app.get("/employee",function (req,res) {
  fs.readFile(__dirname+"/emp.json",function (err,data) {
    res.send(data.toString());
  });
});
app.listen(90);
console.log("Server Listening the Port No.90");
```

Example_4

Reading the data from the mysql database by using NodeJS.

Directory Structure

Node_04

Package.json

Server.js

Package.json

```
{
  "name": "ExpressApp",
  "version": "0.0.1",
  "description": "A Node.js App using express",
  "dependencies": {
    "express": "^4.14.0",
    "mysql": "~2.13.0"
  },
  "engine": "node >=0.6.x"
}
```

Server.js

```
var express=require("express");
var mysql=require("mysql");
var connection = mysql.createConnection({
  host:'localhost',
  user:'root',
  password:'root',
  database:'courses'
});
connection.connect();
connection.query("select * from subjects",
  function(err,records,fields){
    console.log(records);
  });
```

Example_5

Reading the Data From MySQL Data Base and Pass to AngularJS with Node Server.

Directory Structure

Node_05

```
  Index.html
  App.js
  Controllers
    Ctrl.js
  Services
    My_service.js
  Package.json
  Server.js
```

Index.html

```
<!DOCTYPE html>
<html ng-app="app" ng-controller="ctrl">
  <button ng-click="clickMe()">Get Employees</button>
  <br><br>
  <table border="3">
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Age</th>
    </tr>
    <tr ng-repeat="x in list">
      <td>{{x.id}}</td>
      <td>{{x.name}}</td>
      <td>{{x.age}}</td>
    </tr>
  </table>
  <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js"></script>
  <script type="text/javascript" src="app.js"></script>
  <script type="text/javascript" src="services/my_service.js"></script>
  <script type="text/javascript" src="controllers/ctrl.js"></script>
</html>
```

Controllers / ctrl.js

```
app.controller("ctrl",ctrl);
ctrl.$inject=["$scope","my_service"];
function ctrl($scope,my_service){
  $scope.clickMe=function(){
    my_service.my_fun()
      .then(function(response){
        $scope.list=response;
      });
  }
};
```

Services/my_service.js

```
app.service("my_service",my_service);
my_service.$inject=["$http"];
function my_service($http) {
  this.my_fun = function(){
```

```
        return $http.get("/list")
            .then(function(response){
                return response.data;
            });
    }
}
```

App.js

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

package.json

```
{
  "name": "ExpressApp",
  "version": "0.0.1",
  "description": "A Node.js App using express",
  "dependencies": {
    "express": "^4.14.0",
    "mysql": "^2.12.0"
  },
  "engine": "node >=0.6.x"
}
```

Server.js

```
//import the express
var express=require("express");

//import the mysql
var mysql=require("mysql");

//Create the Server Object
var app=express();

//AngularApp - Node App
app.use(express.static(__dirname+"/../Node_05"));

//Default Output
app.get("/",function(req,res){
    res.redirect("/index.html")
});

//Connect to DB
var connection=mysql.createConnection({
    host:'localhost',
    user:'root',
```

```
        password:'root',
        database:'node'
    });
    connection.connect();
    //Create the "get" restful url
    app.get("/list",function(req,res){
        connection.query("select * from emp",
            function(err,records,fields){
                res.send(records);
            });
    });

    //Assign the Port.
    app.listen(3000);
    console.log("Server Listening the Port No.3000");
```

Example_6:

Reading the Data from MongoDB and Passing to Angular Application by using NodeJS

Create the DataBase in MongoDB.

Use mini_project;

Create the Collection in MongoDB

`Db.createCollection("emp")`

Insert the Data into emp.

`Db.emp.insert({"id":1,"name":"Hello_1","age":20})`

`Db.emp.insert({"id":2,"name":"Hello_2","age":22})`

Query the emp collection.

`Db.emp.find()`

Server.js

Package.json

Routes

Feedback.js

Package.json

-

```
{
  "name": "firstpackage",
  "version": "1.0.0",
  "description": "This is our first package",
  "main": "firstServer.js",
  "dependencies": {
    "body-parser": "~1.17.1",
    "express": "^4.14.0",
    "mongodb": "^2.2.24"
  }
}
```

Server.js

```
var express = require("express");
var bodyparser = require("body-parser");
var app = express();
app.use(bodyparser.json());
app.use(express.static(__dirname+"/../POC"));
app.get("/",function (req,res) {
  res.redirect("/index.html");
});
```

```
app.post("/api/login",function (req,res) {
  var userDetails={uname:"admin",upwd:"admin"};
  if(req.body.u_name==userDetails.uname && req.body.u_pwd==userDetails.upwd){
    res.json({login:"success"});
  }else{
    res.json({login:"failure"});
  }
});
var info = require("./routes/info");
app.use("/info",info);

var about = require("./routes/about");
app.use("/about",about);

var feedback=require("./routes/feedback");
app.use("/feedback",feedback);

app.listen(8080);
console.log("Server Listening the Port No.8080");
```

routes/feedback.js

```
var express = require("express");
var mongodb = require("mongodb");
var router = express.Router();
var MongoClient = mongodb.MongoClient;
var url = "mongodb://localhost:27017/mini_project"
router.get("/", function(req, res){
  var response = {};
  MongoClient.connect(url, function(err, db){
    var collection = db.collection("emp");
    collection.find().toArray(function(err, doc){
      response.empDetails = doc;
      console.log(response);
      res.send(response);
    });
    db.close();
  });
});
module.exports=router;
```

Package.json

```
{
  "name": "ExpressApp",
  "version": "0.0.1",
  "description": "A Node.js App using express",
  "dependencies": {
    "body-parser": "^1.15.2",
    "connect": "^3.5.0",
    "express": "^4.14.0"
  },
  "engine": "node >=0.6.x"
}
```

Server.js:

```
var express=require('express');
var app=express();
var bodyparser=require('body-parser');
app.use(bodyparser.json());
app.use(express.static(__dirname+'../spaExample_9'));
app.get('/',function (req,res) {
  res.redirect('/index.html');
});
app.post('/api/login',function (req,res) {
  var uname=req.body.u_name;
  var pwd=req.body.u_pwd;
  if(uname=='admin' && pwd=='admin'){
    res.send({Login:'Success'});
  }else{
    res.send({Login:'Failure'});
  }
});
app.listen(90);
console.log('Server Started on Port No.90');
```

index.html:

```
<!DOCTYPE html>
<html lang="en" ng-app="app">
  <div ui-view>
  </div>
  <script src="lib/angular.min.js"></script>
  <script src="lib/angular-ui-router.min.js"></script>
```



```
<script src="app.js"></script>
<script src="config.js"></script>
<script src="services/myservice.js"></script>
<script src="controllers/login.js"></script>
<script src="controllers/dashboard.js"></script>
</html>
```

App.js:

```
var app=angular.module('app',['ui.router']);
```

config.js:

```
app.config(config);
config.$inject=['$stateProvider','$urlRouterProvider'];
function config($stateProvider,$urlRouterProvider) {
    $urlRouterProvider.otherwise('/login');
    $stateProvider.state('login',{
        url:'/login',
        templateUrl:'templates/login.html',
        controller:'login'
    })
    .state('dashboard',{
        url:'/dashboard',
        templateUrl:'templates/dashboard.html',
        controller:'dashboard'
    });
}
```

Templates/login.html:

```
<!DOCTYPE html>
<html lang="en">
    <label>Uname.</label><input type="text" ng-model="uname"> <br><br>
    <label>Pwd.</label><input type="password" ng-model="pwd"> <br><br>
    <button ng-click="login({u_name:uname,u_pwd:pwd})"><b>Login</b></button>
</html>
```

Templates/dashboard.html:

```
<!DOCTYPE html>
<html lang="en">
    {{var_two}}
</html>
```

Controllers/login.js:

```
app.controller('login',login);
login.$inject=['$scope','$http','$location'];
function login($scope,$http,$location) {
    $scope.login=function (data) {
        $http.post('/api/login',data).
            then(function (response) {
                if(response.data.Login=='Success'){
                    $location.path('dashboard');
                }
            });
    };
};
```

Controllers/dashboard.js:

```
app.controller('dashboard',dashboard);
dashboard.$inject=['$scope'];
function dashboard($scope) {
    $scope.var_two='I am from DashBoard';
}
```

Services/myservice.js

```
aapp.service('myservice',myservice);
function myservice() {
    var obj=this;
    obj.login=function (data) {
        console.log(data.u_name+"...."+data.u_pwd);
    };
    return obj;
}
```

Testing in AngularJS

JavaScript and unit testing:

What are the unit testing options when JavaScript is concerned? To start we need two things - test runner and assertion library. [This](#) StackOverflow question provides decent overview on what's available. It turns out all we need is [Jasmine](#) which is both test runner and [BDD](#) framework, supporting BDD-style of writing tests (or rather *specs*).

Installing Jasmine on Windows

1. Download and install [node.js](#) (it comes as standard Windows .msi installer)
2. Once it's done, type the following in the command line to see whether node's package manager (npm) was successfully installed (we'll use npm to download further modules):
> npm --version
2.5.1

Now we only need few more modules: [Yeoman, Bower and Generator-Jasmine](#). Type following in console:

```
> npm install -g yo  
> npm install -g bower  
> npm install -g generator-jasmine
```

The -g switch tells npm to install packages in node's global modules directory (rather than locally within your project's directories).

To finalize testing environment setup, we need to *scaffold* Jasmine's tests directory. To do that, we'll navigate to project directory and use Yeoman's yo tool:

```
> yo jasmine
```

This will create test directory with index.html and spec/test.js files, which will be of your primary interest.

Running first test

The index.html is Jasmine's *test runner* – open it in browser and your tests will run. “How? What tests?” you might ask. Let's take a quick look and index.html:

```
<!-- include source files here... -->
```

```
<!-- include spec files here... -->
```

```
<script src="spec/test.js"></script>
```

We simply need to reference our implementation and test files:

```
<!-- include source files here... -->
```

```
<script src="../regex-highlighter.js"></script>
```

```
<!-- include spec files here... -->
```

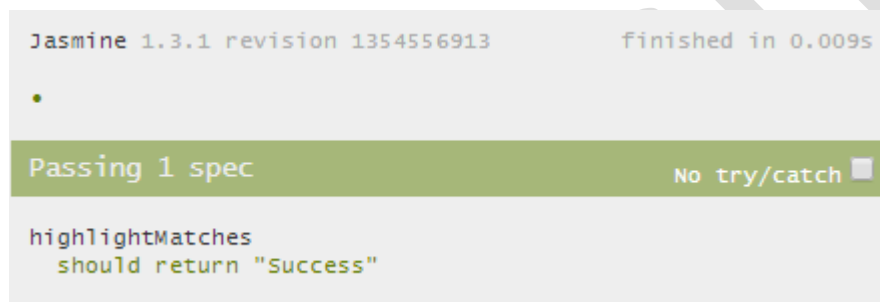
```
<script src="spec/regex-highlighter-tests.js"></script>
```

What's next? First test, obviously. Since this is super-fresh environment our first test for `highlightMatches` function is going to be trivial, requiring the implementation to only return value:

```
'use strict';
```

```
(function () {  
  describe('highlightMatches', function () {  
    it('should return "Success"', function () {  
      expect(highlightMatches('x', 'y')).toBe('Success');  
    });  
  });  
})();
```

Explanation of Jasmine's methods and BDD-style can be found at [Jasmine Introduction page](#). Without further due, we add equally simple implementation of `highlightMatcher` function, refresh `index.html` and Jasmine is happy to announce that our first JavaScript test is very successful one:



Introducing Karma:

Our current setup is up and working and we might just as well be done here. But there is one more thing that will help us greatly when developing JavaScript code – [Karma](#). It is a test runner which watches over our files and runs all tests whenever we make any changes in source files. Perfect match for TDD/BDD environment! You can view introductory video at [Youtube \(14:51\)](#) (don't get confused – tutorial talks about *Testacular*, which was Karma's original name while ago).

To get it we need to execute the following (the karma-cli is Karma's *command line interface* module):

```
> npm install -g karma  
> npm install -g karma-cli
```

Next, navigate to project directory and initialize configuration. Karma will "ask" few simple questions and basing on your answers it will generate config file (*js.config.js*):

```
> karma init jk.config.js
```

Which testing framework do you want to use ?

Press tab to list possible options. Enter to move to the next question.

> jasmine

What is the location of your source and test files ?

You can use glob patterns, eg. "js/*.js" or "test/**/*.Spec.js".

Enter empty string to move to the next question.

> ../regex-highlighter.js

> spec/regex-highlighter-tests.js

Configuration is ready. All that's left to do is simply run Karma, passing configuration file name as argument:

> karma start jk.config.js

Everything should be find and we'll be greeted with message similar to the one below:

```
INFO [karma]: Karma v0.12.31 server started at http://localhost:9876/
INFO [launcher]: Starting browser Chrome
INFO [Chrome 40.0.2214 <Windows 8.1>]: Connected on socket G20E6gvxCi_2PRgKnoip
with id 81325651
Chrome 40.0.2214 <Windows 8.1>: Executed 1 of 1 SUCCESS <0.012 secs / 0.003 secs>
```

Modifying test to make it fail will get noticed immediately:

```
INFO [watcher]: Changed file "D:/NET Workspace/jimmykeen.github.io/scripts/test
s/regex-highlighter-tests.js".
Chrome 40.0.2214 <Windows 8.1> highlightMatches should return "Success" FAILED
Expected 'Success' to be 'Success'.
    at Object.<anonymous> (D:/NETWorkspace/jimmykeen.github.io/scrip
ts/tests/regex-highlighter-tests.js:6:46)
Chrome 40.0.2214 <Windows 8.1>: Executed 1 of 1 <1 FAILED> <0 secs / 0.007 secs>
Chrome 40.0.2214 <Windows 8.1>: Executed 1 of 1 <1 FAILED> ERROR <0.013 secs / 0
.007 secs>
```

Summary:

```
var request = require('request');
it("should respond with hello world", function(done) {
  request("http://localhost:3000/hello", function(error, response, body){
    expect(body).toEqual("hello world");
    done();
  });
});
var request = require('request');
it("should respond with hello world", function(done) {
  request("http://localhost:3000/hello", function(error, response, body){
    done();
  });
}, 250);
var request = require('request');
jasmine.getEnv().defaultTimeoutInterval = 500;
it("should respond with hello world", function(done) {
```

```
request("http://localhost:3000/hello", function(error, response, body){
  done();
}); // timeout after 500 ms
});
```

toBeDefined / toBeUndefined:

If you just want to make sure a variable or property is defined, there's a matcher for that.

There's also one to confirm that a variable or property is `undefined`.

```
it("is defined", function () {
  var name = "Andrew";
  expect(name).toBeDefined();
})
it("is not defined", function () {
  var name;
  expect(name).toBeUndefined();
});
```

toBeTruthy / toBeFalsy:

If something should be true or false, these matchers will do it.

```
it("is true", function () {
  expect(Lib.isAWEEKDay()).toBeTruthy();
});
it("is false", function () {
  expect(Lib.finishedQuiz).toBeFalsy();
});
```

toBeLessThan / toBeGreaterThan:

For all you number people. You know how these work:

```
it("is less than 10", function () {
  expect(5).toBeLessThan(10);
});
it("is greater than 10", function () {
  expect(20).toBeGreaterThan(10);
});
```

toMatch

Have some output text that should match a regular expression? The `toMatch` matcher is ready and willing.

```
it("outputs the right text", function () {  
    expect(cart.total()).toMatch(/\/$\\d*\\.\\d\\d/);  
});
```

toContain:

This one is pretty useful. It checks to see if an array or string contains an item or substring.

```
it("should contain oranges", function () {  
    expect(["apples", "oranges", "pears"]).toContain("orange");  
});
```

There are a few other matchers, too, that you can find in [the wiki](#). But what if you want a matcher that doesn't exist? Really, you should be able to do just about anything with some set-up code and the matchers Jasmine provides, but sometimes it's nicer to abstract some of that logic to have a more readable test. Serendipitously (well, actually not), Jasmine allows us to create our own matchers. But to do this, we'll need to learn a little something else first.

Step 5: Covering Before and After

Often—when testing a code base—you'll want to perform a few lines of set-up code for every test in a series. It would be painful and verbose to have to copy that for every `it` call, so Jasmine has a handy little feature that allows us to designate code to run before or after each test. **Let's see how this works:**

```
describe("MyObject", function () {  
    var obj = new MyObject();  
    beforeEach(function () {  
        obj.setState("clean");  
    });  
    it("changes state", function () {  
        obj.setState("dirty");  
        expect(obj.getState()).toEqual("dirty");  
    })  
    it("adds states", function () {  
        obj.addState("packaged");
```

```
    expect(obj.getState()).toEqual(["clean", "packaged"]);  
  })  
});
```

In this contrived example, you can see how, before each test is run, the state of `obj` is set to “clean”. If we didn’t do this, the changed made to an object in a previous test persist to the next test by default. Of course, we could also do something similar with the `AfterEach` function:

```
describe("MyObject", function () {  
  var obj = new MyObject("clean"); // sets initial state  
  afterEach(function () {  
    obj.setState("clean");  
  });  
  it("changes state", function () {  
    obj.setState("dirty");  
    expect(obj.getState()).toEqual("dirty");  
  })  
  it("adds states", function () {  
    obj.addState("packaged");  
    expect(obj.getState()).toEqual(["clean", "packaged"]);  
  })  
});
```

Here, we’re setting up the object to begin with, and then having it corrected *after every* test. If you want the `MyObject` function so you can give this code a try, [you can get it here in a GitHub gist](#).

Advertisement

Step 6: Writing Custom Matchers

Like we said earlier, customer matchers would probably be helpful at times. So let's write one. We can add a matcher in either a `BeforeEach` call or an `it` call (well, I guess you *could* do it in an `AfterEach` call, but that wouldn't make much sense). Here's how you start:

```
beforeEach(function () {  
  this.addMatchers({  
  });  
});
```

Pretty simple, eh? We call `this.addMatchers`, passing it an object parameter. Every key in this object will become a matcher's name, and the associated function (the value) will be how it is run. Let's say we want to create a matcher that with check to see if one number is between two others. Here's what you'd write:

```
beforeEach(function () {  
  this.addMatchers({  
    toBeBetween: function (rangeFloor, rangeCeiling) {  
      if (rangeFloor > rangeCeiling) {  
        var temp = rangeFloor;  
        rangeFloor = rangeCeiling;  
        rangeCeiling = temp;  
      }  
      return this.actual > rangeFloor && this.actual < rangeCeiling;  
    }  
  });  
});
```

We simply take two parameters, make sure the first one is smaller than the second, and return a boolean statement that evaluates to true if our conditions are met. The important thing to notice here is how we get a hold of the value that was passed to the `expect` function: `this.actual`.

```
it("is between 5 and 30", function () {
```

```
    expect(10).toBeBetween(5, 30);  
  });  
  it("is between 30 and 500", function () {  
    expect(100).toBeBetween(500, 30);  
  });
```

This is what the `SpecHelper.js` file does; it has a `beforeEach` call that adds the matcher `tobePlaying()`. Check it out!

Conclusion: Having Fun Yourself!

There's a lot more you can do with Jasmine: function-related matchers, spies, asynchronous specs, and more. I recommend you [explore the wiki](#) if you're interested. There are also a few accompanying libraries that make testing in the DOM easier: [Jasmine-jQuery](#), and [Jasmine-fixture](#) (which depends on Jasmine-jQuery).

So if you aren't testing your JavaScript so far, now is an excellent time to start. As we've seen, Jasmine's fast and simple syntax makes testing pretty simple. There's just no reason for you not to do it, now, is there?

```
angular.module('controllers', [])  
  .controller('FirstController', ['$scope', 'dataSvc', function($scope, dataSvc) {  
    $scope.saveData = function () {  
      dataSvc.save($scope.bookDetails).then(function (result) {  
        $scope.bookDetails = {};  
        $scope.bookForm.$setPristine();  
      });  
    };  
    $scope.numberPattern = /^\\d*$/;  
  });  
  beforeEach(inject(function($controller){  
    secondController = $controller('SecondController', {  
      dataSvc: mockDataSvc  
    });  
  }));  
  it('should have set pattern to match numbers', function(){  
    expect(secondController.numberPattern).toBeDefined();  
    expect(secondController.numberPattern.test("100")).toBe(true);  
    expect(secondController.numberPattern.test("100aa")).toBe(false);  
  });
```

GRUNT

Grunt is a JavaScript Task Runner which can be used as a command line tool for JavaScript objects. It is a task manager written on top of NodeJS. This tutorial explains how to use GruntJS to automate the build and deployment process in simple and easy steps.

Why Use Grunt?

Grunt can perform repetitive tasks very easily, such as compilation, unit testing, minifying files, running tests, etc.

Grunt includes built-in tasks that extend the functionality of your plugins and scripts.

The ecosystem of Grunt is huge; you can automate anything with very less effort.

History:

The first lines of source code were added to GruntJS in 2011. The Grunt v0.4 was released on February 18, 2013. The Grunt v0.4.5 was released on May 12, 2014. The stable version of Grunt is 1.0.0 rc1 which was released on February 11, 2016.

Advantages:

1. Using Grunt, you can perform minification, compilation, and testing of files easily.
2. Grunt unifies the workflows of web developers.
3. You can easily work with a new codebase using Grunt because it contains less infrastructure.
4. It speeds up the development workflow and enhances the performance of projects.

Disadvantages:

1. Whenever npm packages are updated, you need to wait until the author of the Grunt updates it.
2. Every task is designed to do a specified work. If you want to extend a specified task, then you need to use some tricks to get the work done.
3. Grunt includes a large number of configuration parameters for individual plugins. Usually, Grunt configuration files are longer in length.
4. Grunt is a JavaScript based task runner which means it can automate repetitive tasks in a workflow and it can be used as a command line tool for JavaScript objects.

Features Of Grunt:

1. Grunt makes the workflow as easy as writing a setup file.
2. You can automate repetitive tasks with minimum effort.
3. Grunt is a popular task runner based on NodeJS. It is flexible and widely adopted.
4. It has a straightforward approach which includes tasks in JS and config in JSON.
5. Grunt minifies JavaScript, CSS files, testing files, compiling CSS preprocessor files

(SASS, LESS), etc.

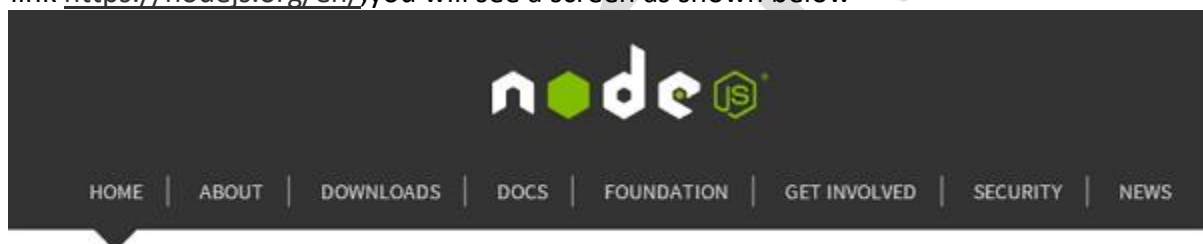
6. Grunt includes built-in tasks that extend the functionality of your plugins and scripts.
7. It speeds up the development workflow and enhances the performance of projects.
8. You can easily work with a new codebase using Grunt because it contains less infrastructure.
9. The ecosystem of Grunt is huge; you can automate anything with very less effort.
10. Grunt reduces the chance of getting errors while performing repetitive tasks.
11. Grunt currently has over 4000 plugins.
12. It can be used in big production sites.

System Requirements for Grunt:

- **Operating System** – Cross-platform
- **Browser Support** – IE (Internet Explorer 8+), Firefox, Google Chrome, Safari, Opera

Installation of Grunt:

Step 1 – We need NodeJs to run Grunt. To download NodeJs, open the link <https://nodejs.org/en/>, you will see a screen as shown below –



Download for Windows (x86)



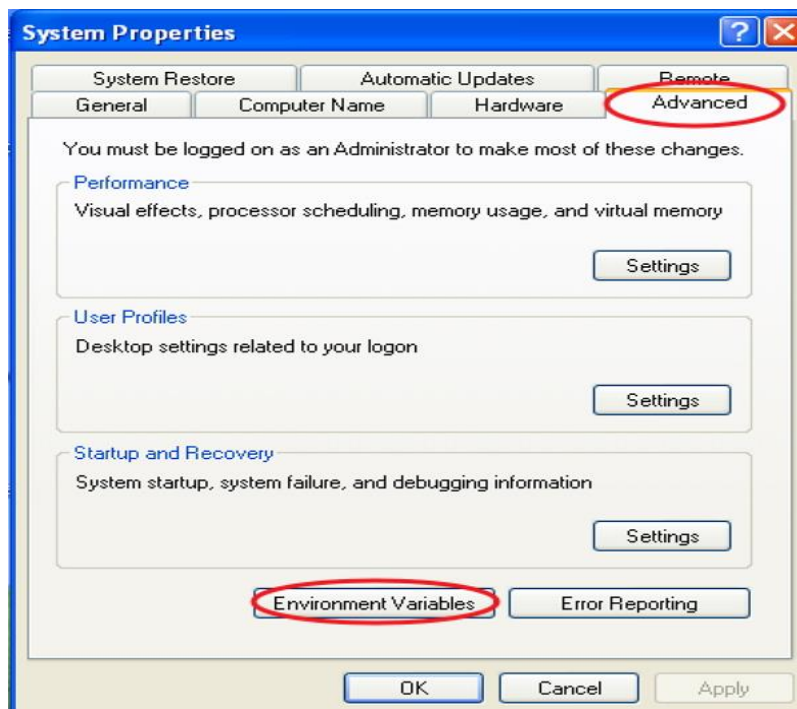
Download the *Latest Features* version of the zip file.

Step 2 – Next, run the setup to install the *NodeJs* on your computer.

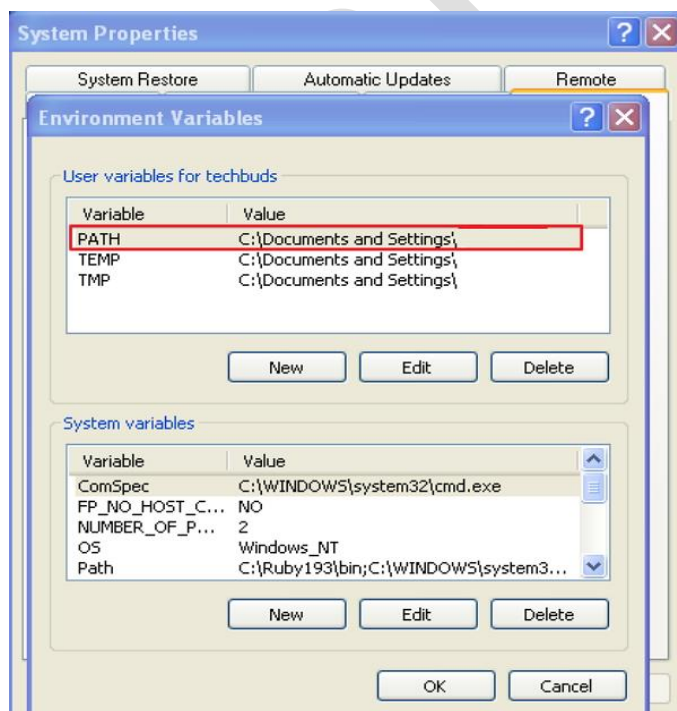
Step 3 – Next, you need to set *environment variables*.

Path User Variable

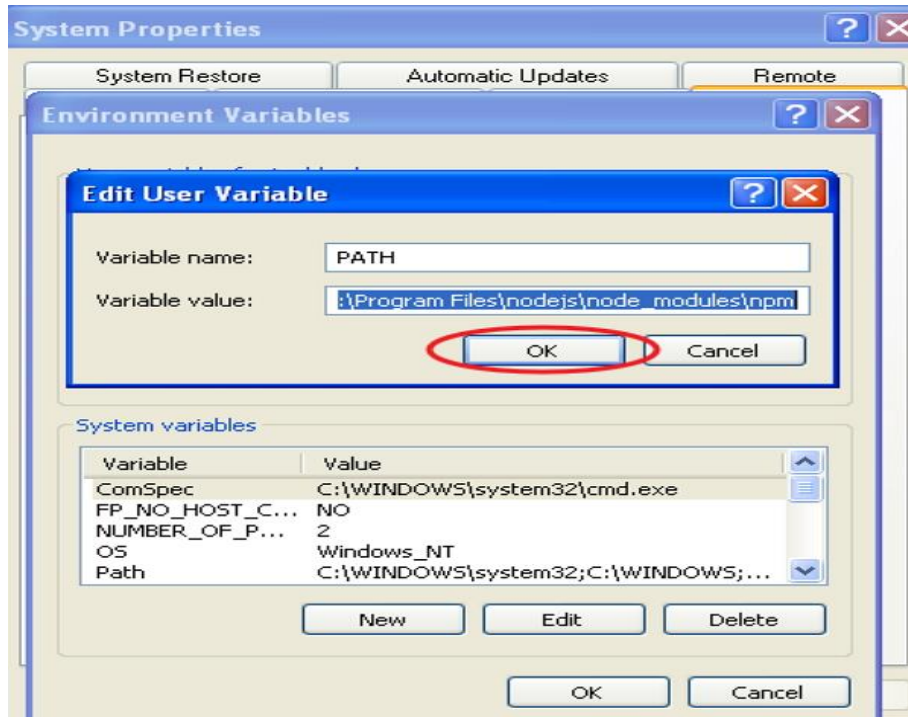
- Right click on **My Computer**.
- Select **Properties**.
- Next, select **Advanced** tab and click on **Environment Variables**.



- Under *Environment Variables* window, double click on the *PATH* as shown in the screen.



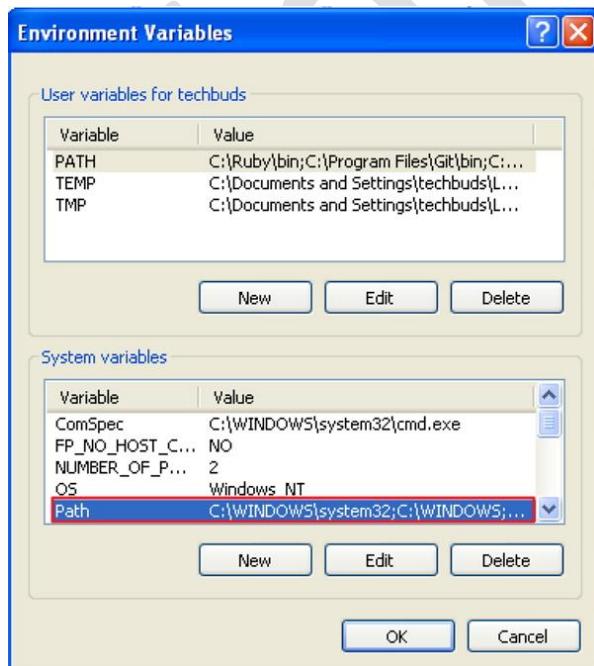
- You will get an *Edit User Variable* window as shown. Add NodeJs folder path in the *Variable Value* field as `C:\Program Files\nodejs\node_modules\npm`. If the path is set already for other files, then you need to put a semicolon(;) after that and add the NodeJs path as shown below –



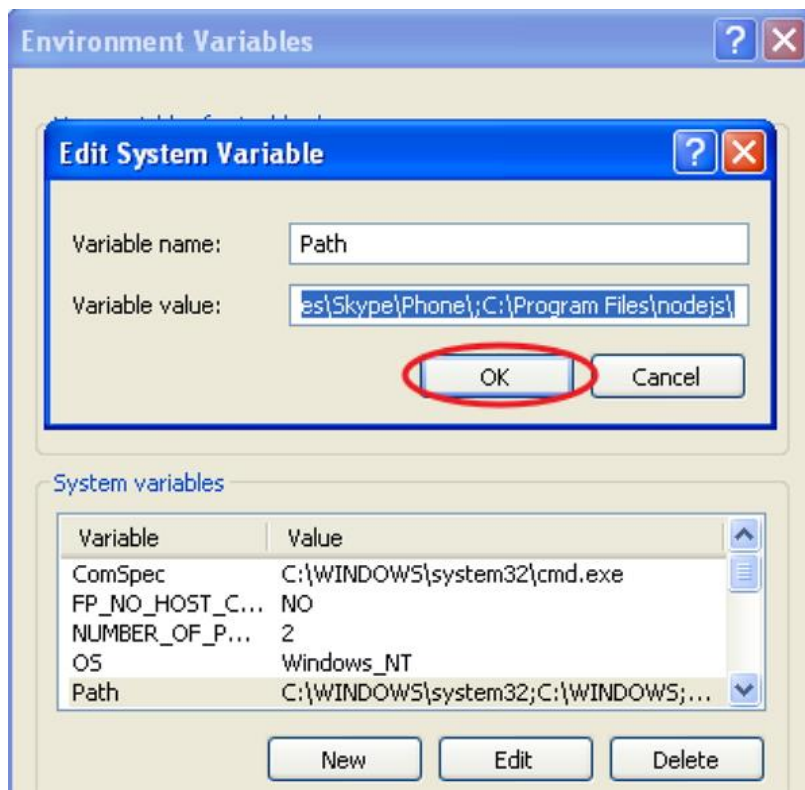
At the end, click the *OK* button.

System Variable:

- Under *System variables*, double click on *Path* as shown in the following screen.



- You will get an *Edit System Variable* window as shown. Add NodeJs folder path in the *Variable Value* field as *C:\Program Files\nodejs* and click *OK* as shown below –



Step 4 – To install grunt on your system you need to install Grunt's command line interface (CLI) globally as shown below –

```
npm install -g grunt-cli
```

Running the above command will put the *grunt* command in your system path, which makes it to run from any directory.

Installing the *grunt-cli* does not install Grunt task runner. The role of the *grunt-cli* is to run the version of Grunt which has been installed next to a *Gruntfile*. It allows a machine to install multiple versions of Grunt simultaneously.

Step 5 – Now, we shall create **configuration files** in order to run Grunt.

package.json:

The *package.json* file is placed in the root directory of the project, beside to the *Gruntfile*. The *package.json* is used to correctly run each listed dependency whenever you run the command **npm install** in the same folder as *package.json*.

The basic *package.json* can be created by typing the following command in the command prompt –


```
npm init
```

The basic *package.json* file will be as shown below –

```
{
  "name": "nareshit",
  "version": "0.1.0",
  "devDependencies": {
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  }
}
```

You can add Grunt and gruntplugins into an existing *pacakge.json* file through the following command –

```
npm install <module> --save-dev
```

In the above command, <module> represents the module to be installed locally. The above command will also add the <module> to *devDependencies* automatically.

For instance, the following command will install latest the latest version of *Grunt* and adds it to your *devDependencies* –

```
npm install grunt --save-dev
```

Gruntfile.js:

The *Gruntfile.js* file is used to define our configuration for Grunt. It is the place where our settings will be written. The basic *Gruntfile.js* file is as shown below –

```
// our wrapper function (required by grunt and its plugins)
// all configuration goes inside this function
module.exports = function(grunt) {
  // CONFIGURE GRUNT
  grunt.initConfig({
    // get the configuration info from package.json file
    // this way we can use things like name and version (pkg.name)
    pkg: grunt.file.readJSON('package.json'),
    // all of our configuration goes here
    uglify: {
```



```
// uglify task configuration
options: {},
build: {}
}
});
// log something
grunt.log.write('Hello world! Welcome to NareshIT!!\n');
// Load the plugin that provides the "uglify" task.
grunt.loadNpmTasks('grunt-contrib-uglify');
// Default task(s).
grunt.registerTask('default', ['uglify']);
};
```

Interview Questions & Answers.

1Q) What is root directive? Can we use angularjs with out ng –app?

* The root directive in angularjs is ng-app. We declare this directive in the root element of the main html document(index.html)

*ng-app is called root directive because the angular framework starts execution of the application from ng-app. And this process is called **Autobootstrap`**

*We can run the angular application with out ng-app through manual bootstrap

Manual bootstrap: Execution of the Angular application forcefully with out ng-app is called Manual bootstrap....

*Sample code to run the application by using manual bootstrap i.e without using ng-app

```
<html>
```

```
<h1 style="color:red" ng-app="app" ng-bind="var_one"></h1>
```

```
<h2 id="myh2" ng-bind="var_two"></h2>
```

```
</html>
```

Manualbootstrap.js

```
angular.element(document).ready(function(response){  
angular.bootstrap(document.getElementById("myh2"),["app"]);  
});
```

In the above code where there is ng-app scope that will be executed using auto bootstrap that is h1 element

And the h2 element executes using manual bootstrap

Note: we have to include(load) the manualbootstrap.js file in the index.html file

Note: we can use any number of **ng-app** in the application.. as the application is divided in to number of modules so we can integrate every modules by creating one more new project each having a ng-app shown below

Example:

In app.js file

```
Var app = angular.module("app",[])
```

Here we can the add different ng-apps of different modules

2Q)Features of Angularjs

Angular js is a Javascript framework to create Rich Internet Applications and the application written in angularjs are cross browser compatible i.e., executes on any major web browser. The most important key features of the Angularjs are

Data-Binding: It is automatic data synchronization between model and view..

Scope: These are objects that refer to the model. They act as a glue between controller and view.

Controller: A controller is a **JavaScript** file which contains attributes/properties and

functions. Each controller has `$scope` as a parameter which refers to the application/module that controller is to control.

Services – AngularJS has several built-in services for example `$http` which is used to make requests using some url and fetch the required data.. These services create the objects only once and this object is shared to all the controllers.. so they are called singleton objects which are instantiated only once in app.

Filters – AngularJS provides filters to transform data

Example: `currency` Format a number to a currency format. `date` Format a date to a specified format. **filter** Select a subset of items from an array. `json` Format an object to a JSON string... These select a subset of items from an array and returns a new array

Directives – directives are markers on a DOM element (such as an attribute, element name or CSS class) that tell AngularJS's **HTML compiler** to attach a specified behavior to that DOM element

There are two types of directives they are

pre-defined directives: These are built-in directives provided by the angular framework
ex: `ng-model`, `ng-bind`, `ng-repeat` etc

custom directives: creating our own directives based on application requirement called as custom directives

ex: `my_directive`..... custom directive

`<my_directive> </my_directive>`.....element level usage

`<div my_directive></div>`attribute level usage

Templates – These are the rendered view with information from the controller and model. These can be a single file (like `index.html`) or multiple views in one page using "partials"

Routing – It is concept of switching views. This concept is in question no :4

Deep Linking – Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

Dependency Injection – AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

3Q) Directives and uses:

Below are some pre-defined directives of AngularJS

Directive Name	Functionality	Example
1, <code>ng-app</code>	It is a root directory in angularJS ,Framework will start the execution from <code>ng-app</code> .	Ex: <code><html ng-app="app"></code>

	This directive can be mention in "HTML"(view).	
2,ng-controller	This directive used to declare controllers. We can have one or more controllers per application. In general we will declare controllers in view	<u>Ex:</u> app. controller("ctrl",ctrl); ctrl.\$inject=["\$scope"]; function ctrl(\$scope){ \$scope.varone="angularJS"; }
3,ng-model	Binding the view into the model, which other directives such as input, text area or select require. Providing validation behavior (i.e. required, number, email, url).	<u>Ex:</u> <form name="testForm" ng-controller="ExampleController"> <input ng-model="val" name="angular"> </form>
4,ng-bind	The ng-bind attribute tells AngularJS to replace the text content of the specified HTML element with the value of a given expression It is preferable to use ng-bind instead of {{ expression }}	<u>Ex:</u>
5,ng-repeat	Used to iterate the list of elements from collection	Ex:<ng-repeat="x in [1,2,3,4]">{{x}}
6,ng-click	The ng-click directive allows you to specify custom behavior when an element is clicked.	Ex: <ng-click ng-click="expression"> ... </ng-click>
7, ng-options	The ng-options attribute can be used to dynamically generate a list of <option> elements for the <select>element	Ex: <select ng-model="my_model" ng-options="x.name as x.id for x in data"></select>
8,ng-init	Used to initialize the application data syntactically. We can initialize the data in the form of key and value	Ex: <ng-init "key1=Hi;key2=Hello">
9, ng-show	The ng-show directive	Ex:

	shows or hides the given HTML element based on the expression provided to the ng-show attribute.	<code><div ng-show="myValue"></div></code>
10,ng-switch	Used to write the "switch cases" and "default cases" in angular application	Ex: <code></code> <code> ...</code> <code> ...</code> <code> <div ng-switch-default>...</div></code> <code></code>
11,ng-if	Used to write the conditions in view	Ex: <code><div ng-repeat='x in data'></code> <code> <div ng-if ='x!= 'hello1' && 'hello5' "></code> <code> {{x}}</code> <code></div></code> <code></div></code>
12,ng-cloak	Used to avoid unparsed data in two way data binding	Ex: <code><div ng-cloak style="color:blue" ng-controller="c1"></code> <code> {{var1}}</code> <code></div></code>
13,ng-class-even ng-class-odd	Used to apply css classes to the even rows of table Used to apply css classes to the odd rows of table	Ex: <code><ng-repeat ="x in data" ng-class-even='even' "></code> <code><ng-repeat ="x in data" ng-class-odd='odd' "></code>
14,ng-include	Used to split the view into number of sub views	<code><div ng-include="view.html"></code> <code></div></code>
15,ng-dblclick	The ng-dblclick directive allows you to specify custom behavior on a double click event.	Ex: <code><button ng-dblclick="count = count + 1" ng-init="count=0"></code> <code> Increment (on double click)</code> <code></button></code> <code>count: {{count}}</code>

16,ng-submit	Used to submit the form to the controller	Ex: <form ng-submit="user defined functions"> //form elements <input type="submit"> </form>
17,ng-mousedown	Whenever we are using down operation automatically framework will execute user defined functions	<div ng-mousedown="down()">
ng-mouseover	Specify custom behavior on mouse over event.	<div ng-mouseover="over()">
ng-mouseleave	Specify custom behavior on mouse leave event.	<div ng-mouseleave="leave()">

4Q) Dependencies for routing

Loading the Target Templates to the Source Templates without refreshing is called as Routing in Single Page Application

In AngularJS Routing can be done in two ways:

- **ngRoute Module**
- **ui.Router Module**

ngRoute Module: The ngRoute Module routes your application to different pages without reloading the entire application.

Ui.Router Module: The UI-Router is a routing framework for AngularJS built by the AngularUI team. It provides a different approach than ngRoute in that it changes your application views based on state of the application and not just the route URL (i.e. ngRoute just route the url and ui router changes the view based on state of application)

Development of Single Page Application By Using ngRoute Module.

- ngRoute is the Predefined Module in AngularJS
- ngRoute is the Native Module in AngularJS

TO Download the ngRoute module by using bower:

bower.json-

dependencies:{

"angular": "~1.5.0",

"angular-route": "~1.5.0"

```
}
```

TO Download the ui-router module by using bower

bower.json -

```
dependencies:{  
  "angular":"~1.6.0",  
  "angular-ui-router":"~0.2.18"  
}
```

Differences Between ngRoute and ui.router module.

- ngRoute module is "native module" in angularJS

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

- ui.router module is the 3rd party module.

var app=angular.module("app",["ui.router"]);

- ngRoute won't supports the Nested Views in Single Page Applications.
- ui.router supports the Nested Views in Single page Application.
- ngRoute won't supports the Named Views in Single Page Application.
- ui.router supports the Named Views in Single page Application.
- ngRoute won't Supports the "Objects passing" as URL Parameters
- ui.router supports the "Objects" as the URL Parameters
- ngRoute in bower

angular-route:'~1.5.0'

- ui.router in bower

angular-ui-router:"~0.2.18"

5Q) What is MVC?

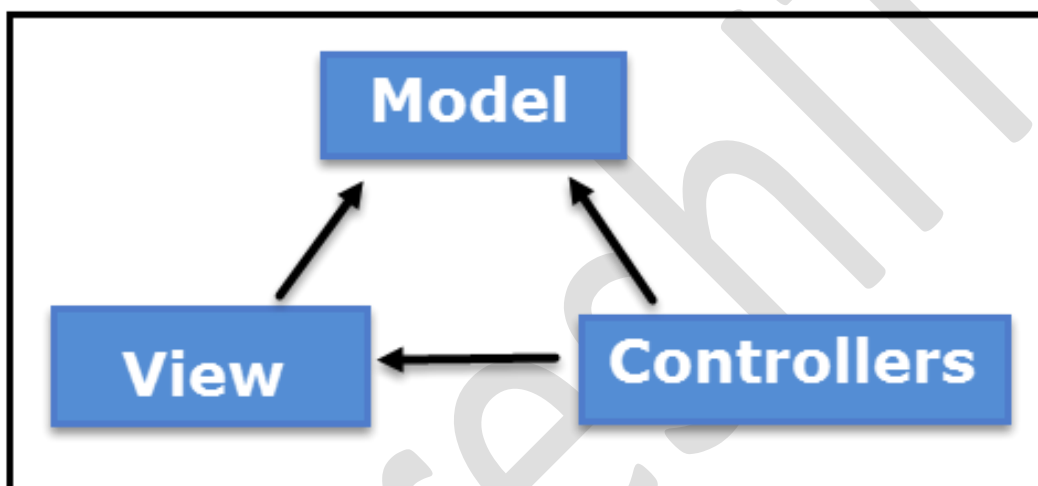
- **MVC is a design pattern used to isolate business logic from presentation.**
- The MVC pattern has been given importance by many developers as a useful pattern for the reuse of object code and a pattern that allows them to reduce the time it takes to develop applications with user interfaces.
- The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

MVC Components:

- **Model:** The Model component corresponds to all the data related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic related data. For example, a

Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

- **View:** The View component is used for all the UI logic of the application. For example, the Customer view would include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.
- **Controller:** Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller would handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller would be used to view the Customer data.



6Q). Ng-init directive

- > ng-init directive is used to initialize the application data Statically.
- > ng-init directive is used to define a local variable with value.
- > It is supported by all html elements.

Syntax:

`<element ng-init="expression">` `</element>`

Example:

```
<div ng-app="app "
  ng-init="name='naresh it' ">
  <h1>{{ name }} </h1>
</div>
```

Note: In this example we are giving 'name' as local variable.

7Q) .Difference between \$scope and \$rootScope.

-> \$scope is a glue between controller. and view.

\$rootScope is a parent object of all \$scope angular objects created in a web page.

->\$scope is created with ng-controller. \$rootScope is created with ng-app.

->Each angular application has exactly one rootscope, but may have several child scopes.

->The \$scope will be available only inside the controller. But we can access \$rootScope in all the controllers.

8Q) What is Internationalization

Internationalization is the process of developing products in such a way that they can be localized for languages and cultures easily. Localization (l10n), is the process of adapting applications and text to enable their usability in a particular cultural or linguistic market.

We implement internationalization in two ways

i18n and l10n

9Q)How to implement Internationalization in Angular Js?

---> To implement Internationalization in angularjs, the angular-translate is an AngularJS module.

--->that brings i18n (internationalization) and l10n (localization) into your Angular app.

--->It allows you to create a JSON file that represents translation data as per language.

--->The angular-translate library (angular-translate.js) also provides built-in directives and filters that make the process of internationalizing simple.

i18n and l10n

First we need to install:

bower install angular-i18n

Example:

```
<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 = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
  <!-- <script src = "https://code.angularjs.org/1.3.14/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:

AngularJS Sample Application
\$100.00
Feb 27, 2017
123.45

10Q) Types of data binding in angularjs?

The **data binding** is the data synchronization processes between the model and view components

In angularjs when model data got changed that time the view data will change automatically and vice versa.

We have two types of data bindings available in angularjs those are

1. One-Way data binding
2. Two-Way data binding

One-Way data binding

- In **One-Way** data binding, view will not update automatically when data model changed.
- we need to write custom code to make it updated every time.
- Its not a synchronization processes and it will process data in one.

Two-way Data Binding

In **Two-way** data binding, view updates automatically when data model changed.

- Its synchronization processes and two way data binding.
- This two-way data binding using [ng-model](#) directive.
- If we use ng-model directive in html control it will update value automatically whenever data got changed in input control.

11. What is manual bootstrap?

If you need to have more control over the initialization process, you can use a manual bootstrapping method instead. Examples of when you'd need to do this include using script loaders or the need to perform an operation before AngularJS compiles a page.

Here is an example of manually initializing AngularJS:

```
<!doctype html>
<html>
<body>
  <div ng-controller="MyController">
    Hello {{greetMe}}!
  </div>
  <script src="http://code.angularjs.org/snapshot/angular.js"></script>
  <script>
    angular.module('myApp', [])
      .controller('MyController', ['$scope', function ($scope) {
        $scope.greetMe = 'World';
      }]);
    angular.element(function() {
      angular.bootstrap(document, ['myApp']);
    });
  </script>
</body>
</html>
```

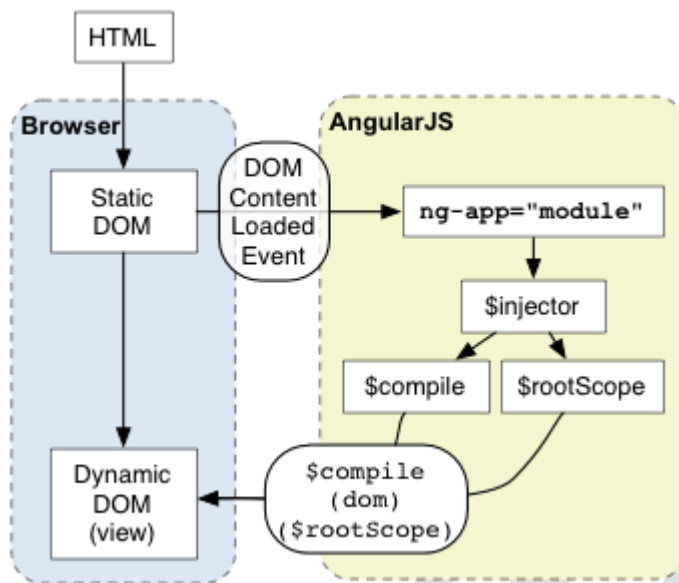
Note that we provided the name of our application module to be loaded into the injector as the second parameter of the [angular.bootstrap](#) function. Notice that angular.bootstrap will not create modules on the fly. You must create any custom [modules](#) before you pass them as a parameter.

You should call `angular.bootstrap()` *after* you've loaded or defined your modules. You cannot add controllers, services, directives, etc after an application bootstraps.

This is the sequence that your code should follow:

1. After the page and all of the code is loaded, find the root element of your AngularJS application, which is typically the root of the document.
2. Call [angular.bootstrap](#) to [compile](#) the element into an executable, bi-directionally bound application.

12. What is Auto bootstrap?



AngularJS initializes automatically upon DOMContentLoaded event or when the angular.js script is evaluated if at that time document.readyState is set to 'complete'. At this point AngularJS looks for the [ngApp](#) directive which designates your application root. If the [ngApp](#) directive is found then AngularJS will:

- load the [module](#) associated with the directive.
- create the application [injector](#)
- compile the DOM treating the [ngApp](#) directive as the root of the compilation. This allows you to tell it to treat only a portion of the DOM as an AngularJS application.

```
</doctype html>
```

```
<html ng-app="optionalModuleName">
```

```
<body>
```

```
  I can add: {{ 1+2 }}.
```

```
  <script src="angular.js"></script>
```

```
</body>
```

```
</html>
```

There are a few things to keep in mind regardless of automatic or manual bootstrapping:

- While it's possible to bootstrap more than one AngularJS application per page, we don't actively test against this scenario. It's possible that you'll run into problems, especially with complex apps, so caution is advised.
- Do not bootstrap your app on an element with a directive that uses [transclusion](#), such as [ngIf](#), [ngInclude](#) and [ngView](#). Doing this misplaces the app [\\$rootElement](#) and the app's [injector](#), causing animations to stop working and making the injector inaccessible from outside the app.

13. Is AngularJS a library, framework, plugin or a browser extension?

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

Definition of AngularJS as put by its [official documentation](#) is as follows –

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology.

Features

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

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

Core Features

Following are most important core features of AngularJS –

- **Data-binding** – It is the automatic synchronization of data between model and view components.
- **Scope** – These are objects that refer to the model. They act as a glue between controller and view.
- **Controller** – These are JavaScript functions that are bound to a particular scope.
- **Services** – AngularJS come with several built-in services for example `$https`: to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
- **Filters** – These select a subset of items from an array and returns a new array.

- **Directives** – Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ngBind, ngModel...)
- **Templates** – These are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".
- **Routing** – It is concept of switching views.
- **Model View Whatever** – MVC is a design pattern for dividing an application into different parts (called Model, View and Controller), each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.
- **Deep Linking** – Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

14. Does AngularJS has dependency on jQuery?

AngularJS has no dependency on jQuery library. But it can be used with jQuery library. If jQuery is available, angular.element is an alias for the [jQuery](#) function. If jQuery is not available, angular.element delegates to AngularJS's built-in subset of jQuery, called "jQuery lite" or **jqLite**.

jqLite is a tiny, API-compatible subset of jQuery that allows AngularJS to manipulate the DOM in a cross-browser compatible way. jqLite implements only the most commonly needed functionality with the goal of having a very small footprint.

To use jQuery, simply ensure it is loaded before the angular.js file. You can also use the [ngJq](#) directive to specify that jqLite should be used over jQuery, or to use a specific version of jQuery if multiple versions exist on the page.

15 .\$watch will watch changes which are done within the scope

The AngularJS \$watch is used to observe properties on a single object and notify you if one of them changes.

In other word watch is shallow watches the properties of an object and fires whenever any of the properties change.

This method is for watching changes of scope variables.

This method has callback function which gets called when the watching properties are changed.

Example:

```
<!doctype html>
<html ng-app="app" ng-controller="ctrl">
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.1/angular.min.js"></script>
<input type="text" ng-model="expression">
<script>
  var app=angular.module("app",[]);
  app.controller("ctrl",ctrl);
  ctrl.$inject=["$scope"]
  function ctrl($scope){
    $scope.$watch("expression",function(
      newValue,oldValue,scope ){
      console.log("newValue="+ newValue);
      console.log("oldValue="+ oldValue);
      console.log(scope.expression);
    });
  }
</script>
</html>
```

16.What is the difference between \$parse and \$eval?**\$eval:**

The AngularJS \$eval method is used to executes the AngularJS expression on the current scope and returns the result.

In AngularJS, expressions are similar to JavaScript code snippets that are usually placed in bindings such as {{ expression }}.

AngularJS used internally \$eval to resolve AngularJS expressions, such as {{variable}}.

The important difference between \$eval and \$parse is that \$eval is a scope method that executes an expression on the current scope, while \$parse is a globally available service. The \$eval method takes AngularJS expression and local variable object as parameter.

\$eval takes an angular expression, evaluates it and returns the actual result of that expression.

for example1:

```
$scope.a = 2;
var result = $scope.$eval('1+1+a');
// result is 4
```

Example2:

```
<!DOCTYPE html>
<html lang="en" ng-app="app" ng-controller="ctrl">
<script src="bower_components/angular/angular.min.js"></script>
<div>
<h1>a={{a}}</h1><br>
<h1>b={{b}}</h1><br>
<h1>Result={{a * b}}</h1>
</div>
  <button ng-click="clickMe()">DEMO </button>
<script>
  var app=angular.module("app",[]);
  app.controller("ctrl",ctrl);
  ctrl.$inject=["$scope"];
  function ctrl($scope) {
    $scope.a = 10;
    $scope.b = 20;
    $scope.emp={
      ename:"rk",age:24
    };
    $scope.clickMe = function () {
      var r = $scope.$eval("a*b")
      alert("result:" + r);
      var r2=$scope.$eval("emp.ename");
      alert(r2);
      var r3=$scope.$eval("a*b*3");
      alert(r3);
      var r4=$scope.$eval("a*b*3*c");
      alert(r4);
      var r5=$scope.$eval("a*b*3*c",{c:5});
      alert(r5);
      var r6=$scope.$eval("a*b*3*c",{a:2,c:5});
      alert(r6);
      var r=$scope.$eval(function($scope,locals){
        return $scope.a*$scope.b
      })
      alert(r);
      var r=$scope.$eval(function($scope,locals){
        return $scope.a*$scope.b*locals.c,
        {a:2,b:3,c:6});
      alert(r);
```



```

    };
  }
</script>
</html>

```

\$parse:

\$parse does not require scope. It takes an expression as a parameter and returns a function.

The function can be invoked with an object that can resolve the locals:

\$parse takes an angular expression and returns a function that represents that expression.

For example1:

```
var fn = $parse('1+1+a');
```

```
var result = fn({ a: 2 });
```

```
// result is 4
```

example2:

```

<!DOCTYPE html>
<html lang="en" ng-app="app" ng-controller="ctrl">
<script src="bower_components/angular/angular.min.js"></script>
<div>
<h1>a={{a}}</h1><br>
<h1>b={{b}}</h1><br>
<h1>Result={{a * b}}</h1>
  {{emp.ename}}
</div>
<button ng-click="clickMe()">parseDemo </button>
<script>
  var app=angular.module("app",[]);
  app.controller("ctrl",ctrl);
  ctrl.$inject=["$scope","$parse"];
  function ctrl($scope,$parse) {
    $scope.a = 10;
    $scope.b = 20;
    $scope.emp = {
      ename: "rk", age: 24
    };
    $scope.clickMe = function () {
//      var my_fun = $parse("a*b"); //return a function
//      var r = my_fun($scope);
//      alert("result=" + r);

```

```

        //alert("result="+ $parse("a*b")($scope)
        // alert("result="+ $parse("a*b")({a:2,b:3}));
//      var my_fun = $parse("a*b"); //return a function
//      var r1 = my_fun($scope);
//      alert("result=" + r1);
//      var r2=my_fun({a:2,b:3});
//      alert("result=" + r2);
// alert($parse("emp.ename")($scope));
// ($parse("emp.ename").assign($scope,"satti"));
// alert($parse("emp.ename")($scope));
    }
}
</script>

```

17.Explain \$scope in angular?

The scope is the binding part between the HTML (view) and the JavaScript (controller).

The scope is an object with the available properties and methods.

The scope is available for both the view and the controller.

How to Use the Scope:

When you make a controller in AngularJS, you pass the \$scope object as an argument:

Example:

Properties made in the controller, can be referred to in the view:

```

<div ng-app="myApp" ng-controller="myCtrl">
<h1>{{carname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.carname = "Volvo";
});
</script>

```

When adding properties to the \$scope object in the controller, the view (HTML) gets access to these properties.

In the view, you do not use the prefix \$scope, you just refer to a propertyname, like {{carname}}.

Understanding the Scope:

If we consider an AngularJS application to consist of:

View, which is the HTML.

Model, which is the data available for the current view.

Controller, which is the JavaScript function that makes/changes/removes/controls the

data.

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

Example:

If you make changes in the view, the model and the controller will be updated:

```
<div ng-app="myApp" ng-controller="myCtrl">
<input ng-model="name">
<h1>My name is {{name}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.name = "Ramakrishna";
});
</script>
```

18) is angularjs code is reusable

I'm using Angular to develop commenting functionality for a web app. Currently there are two sections in the application where a user can comment:

Category

Product

About 90% of the commenting functionality is the same for both sections and as such I would like to make this reusable - i.e write some service or controller that I can reference/use as a base.

So far, my research seems to point to using a factory service but unfortunately this doesn't seem to work (I've spent the whole day running through various tutorials).

It is quite possible that I am over thinking this and making it far too complicated but I honestly don't know which way to turn anymore.

Herewith a quick and dirty overview of what I have so far:

HTML view for the category

Controller for the category (receives data from service and posts data to service in order to bind data to model)

Service for the category (retrieve and stores all the necessary data)

The product uses the same logic and a lot of the code in the service and controller will be duplicated. I've merged the two services into one service successfully but I'm having trouble doing the same for the controller.

Do I:

Write a base controller that will communicate with the above mentioned service and that will hookup with the two existing controllers

OR

Write a factory/provider service that hooks up to the two existing controllers as well as the above mentioned service.

19) in our programming we are writing service as function that is that is to achieve reusability.

Yes

20) what is the difference between \$watch and \$observe

\$observe() is a method on the Attributes object, and as such, it can only be used to observe/watch the value change of a DOM attribute.

It is only used/called inside directives. Use \$observe when you need to observe/watch a DOM attribute that contains interpolation (i.e., {{}}'s).

E.g., attr1="Name: {{name}}", then in a directive: attrs.\$observe('attr1', ...).

(If you try scope.\$watch(attrs.attr1, ...) it won't work because of the {{}}s -- you'll get undefined.) Use \$watch for everything else.

\$watch() is more complicated. It can observe/watch an "expression", where the expression can be either a function or a string.

If the expression is a string, it is \$parse'd (i.e., evaluated as an Angular expression) into a function.

(It is this function that is called every digest cycle.) The string expression can not contain {{}}'s.

\$watch is a method on the Scope object, so it can be used/called wherever you have access to a scope object, hence in

- # a controller -- any controller -- one created via ng-view, ng-controller, or a directive controller

- # a linking function in a directive, since this has access to a scope as well

Because strings are evaluated as Angular expressions, \$watch is often used when you want to observe/watch a model/scope property.

E.g., attr1="myModel.some_prop", then in a controller or link function:

scope.\$watch('myModel.some_prop', ...) or

scope.\$watch(attrs.attr1, ...) (or scope.\$watch(attrs['attr1'], ...)).

(If you try attrs.\$observe('attr1') you'll get the string myModel.some_prop, which is probably not what you want.)

As discussed in comments on @PrimosK's answer, all \$observes and \$watches are checked every digest cycle.

Directives with isolate scopes are more complicated. If the '@' syntax is used, you can \$observe or \$watch a DOM attribute that contains interpolation (i.e., {{}}'s).

(The reason it works with \$watch is because the '@' syntax does the interpolation for us, hence \$watch sees a string without {}'s.)

To make it easier to remember which to use when, I suggest using \$observe for this case also.

To help test all of this, I wrote a Plunker that defines two directives. One (d1) does not create a new scope, the other (d2) creates an isolate scope.

Each directive has the same six attributes. Each attribute is both \$observe'd and \$watch'ed.

```
<div d1 attr1="{{prop1}}-test" attr2="prop2" attr3="33" attr4="a_string"
      attr5="a_string" attr6="{{1+aNumber}}"></div>
```

Look at the console log to see the differences between \$observe and \$watch in the linking function.

Then click the link and see which \$observes and \$watches are triggered by the property changes made by the click handler.

Notice that when the link function runs, any attributes that contain {}'s are not evaluated yet (so if you try to examine the attributes, you'll get undefined).

The only way to see the interpolated values is to use \$observe (or \$watch if using an isolate scope with '@').

- #\$observe is used in linking function of directives.

- #\$watch is used on scope to watch any changing in its values.

- Keep in mind : both the function has two arguments,

\$observe/\$watch(value : string, callback : function);

value : is always a string reference to the watched element (the name of a scope's variable or the name of the directive's attribute to be watched)

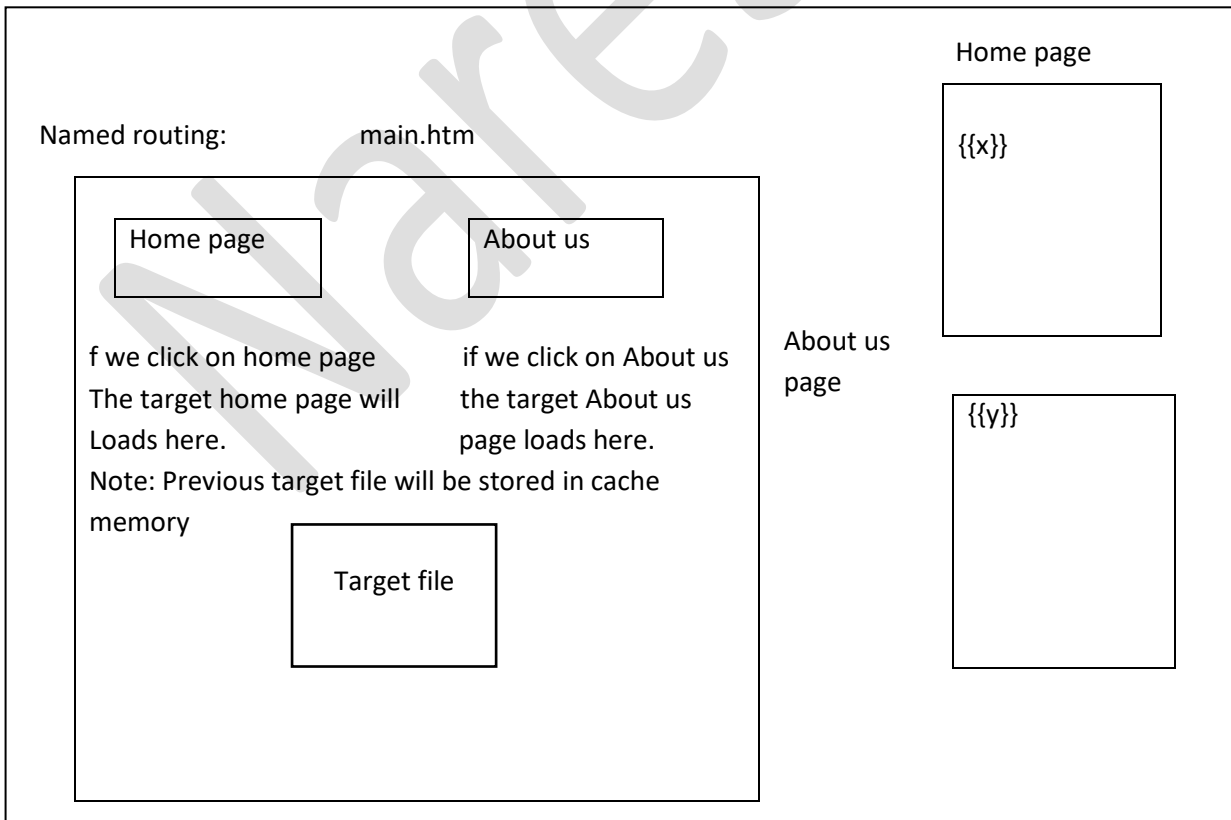
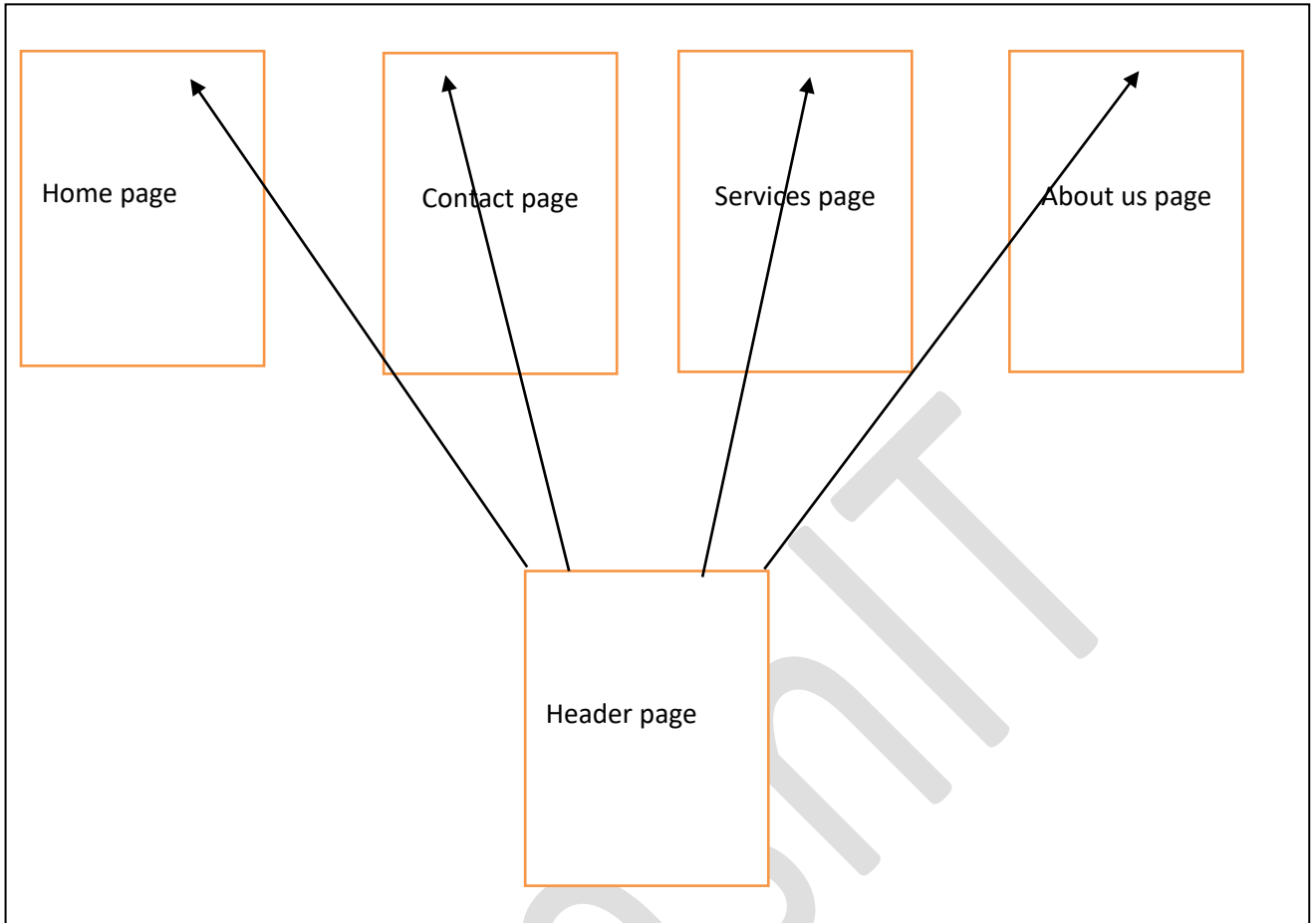
callback : the function to be executed of the form function (oldValue, newValue)

Therefore, getting the values of these attributes is an asynchronous operation. (And this is why we need the \$observe and \$watch functions.)

21. Differentiate between named routing and custom directives?

Custom directives	Routing
<p>1. Creating your own directive Based on application requirement is called custom directive.</p> <p>2. We can achieve custom directives in three levels. Element level. Attribute level. CSS class level.</p> <p>3. Here we don't need any objects or services to use above levels.</p> <p>4. Most advantage is to reuse the custom directive in the place where ever we need.</p> <p>5. ng-controller directive is the connection between template(vie w) and its controller(model).</p>	<p>1. Loading the target templates without refreshing web page is called as routing in single page application.</p> <p>2. We can achieve named routing in two ways. Ng-route. Ui-router.</p> <p>3. Here we need to pass objects to use the services like\$route Provider, \$state provider.</p> <p>4. There is no necessity of reusing routing because at the time of loading web pages in the browser it also loads its resources like html, css, js files etc without refreshing the source content.</p> <p>5.Mapping with key and value pairs is the connection between target(view) and its controllers(model), This is done in sconfiguration file, Mapping will done before launching web pages.</p>

Custom directive scenario: Instead of coding common part in every view. Let's create a separate html file for common part and declare it as custom directive, then use it as custom directive in every html file which Leads to less time consuming, better understanding, increases reusability as it is shown below.



22. List at least three ways to communicate between modules of your application using angular js functionality?

A module can be injected into another module, in which case the container module has access to all elements of the injected module. If you look at angular seed project, modules are created for directive, controllers, filters etc, something like this `angular.module("myApp", ["myApp.filters", "myApp.services", "myApp.directives", "myApp.controllers"])` This is more of a re usability mechanism rather than communication mechanism.

The second option is would be to use services. Since services are singleton and can be injected into any controller, they can act as a communication mechanism.

Third option is to use parent controller using `$scope` object as it is available to all child controllers.

You can also inject `$rootScope` into controllers that need to interact and use the `$broadcast` and `$on` methods to create a service bus pattern where controllers interact using pub\sub mechanism.

23. Different types of errors in Angular js? Can we do or create custom errors?

A. Errors:

Some Namespace errors are given below.

ng- this is the error in ng-namespace.

Name	Description
badident	Invalid identifier expression
dupes	Duplicate Key in Repeater
iexp	Invalid Expression
iidexp	Invalid Identifier

Ng-repeat- list of errors in the ngRepeat namespace.

areq	Bad Argument
badname	Bad `hasOwnProperty` Name
btstrpd	App Already Bootstrapped with this Element
cpi	Bad Copy
cpta	Copying TypedArray

cpws	Copying Window or Scope
test	Testability Not Found

Ng-model: list of errors in the ngModel namespace.

Name	Description
constexpr	Non-Constant Expression
datefmt	Model is not a date object
nonassign	Non-Assignable Expression
nopromise	No promise
numfmt	Model is not of type `number`

Ng-options: list of errors in the ngOptions namespace.

Name Description

iexp	Invalid Expression
----------------------	--------------------

\$q: list of errors in the \$q namespace.

Name Description

norslvr	No resolver function passed to \$Q
gcycle	Cannot resolve a promise with itself

The TypeError object represents an error when a value is not of the expected type.

Uncaught reference error: //didn't got the answer.

Custom errors:

\$error: Is an object hash, containing references to all invalid controls or forms, where:

keys are validation tokens (error names)

values are arrays of controls or forms that are invalid with given error.

24.what is the difference between ngshow and ngif and Ng hide?

ng-Show/nghide

==>ngShow directive is a part of module ng.

==>It is used to show or hide the elements based on boolean values.

==>The ng-show directive shows the specified HTML element if the expression evaluates to true, otherwise the HTML element is hidden.

==>ngShow/ngHide work with the show and hide events that are triggered when the directive expression is true and false in Animations

==>The great part about these directives is that "we don't have to do any of the showing or hiding ourselves with CSS or JavaScript".

==>Supported by all HTML elements.

==>Both ng-show and ng-if receive a condition and hide from view the directive's element in case the condition evaluates to false.

==>ng-show (and its sibling ng-hide) toggle the appearance of the element by adding the CSS display: none style.

==>The ng-hide directive hides the HTML element if the expression evaluates to true.

==>ng-hide is also a predefined CSS class in AngularJS, and sets the element's display to none.

Syntax:

`<element ng-show="expression"></element>`

Parameter:

Value	Description
expression	An expression that will show the element only if the expression returns true.

Example:

`<!DOCTYPE html>`
`<html>`
`<body ng-app="">`
Show HTML: <input type="checkbox" ng-model="myVar">
`<div ng-show="myVar">`
`<h1>Welcome</h1>`
`<p>Welcome to my Show Directive.</p>`
`</div>`
`</body>`
`</html>`

25.How to implement securities in angular js? And what do u mean by IAutho?

Security

==>Security is one of the most important parts of writing a web application—perhaps the most important part!

==>Angular is not made to enhance security of our web application but to help the web application run smooth and user-friendly.

==>Do not mix client and server templates

==>Do not use user input to generate templates dynamically

==>Do not run user input through `$scope.$eval` (or any of the other expression parsing functions)

==>You can not access a angular variable/scope from console of your browser .

==>It Prevents cross-side-scripting attacks.

==>Prevents HTML injection attacks.

==>NG-CSP directive is given in angular to stop injecting inline codes to the application.

==>you have to send every http using ajax and you need to have an api for the back-end.

==>When developers build client apps with server back ends they approach the application as though they control the entire ecosystem.

==>Assumptions are often made that the client they built will only ever talk to the server side APIs they built in the way they designed them.

==>HTTP Requests, Whenever your application makes requests to a server there are potential security issues that need to be blocked.

==>Using Local Caches, There are various places that the browser can store (or cache) data.

==>Within AngularJS there are objects created by the `$cacheFactory`. These objects, such as `$templateCache` are used to store and retrieve data,

==>primarily used by `$http` and the `script` directive to cache templates and other data.

==>Similarly the browser itself offers `localStorage` and `sessionStorage` objects for caching data.

==>Attackers with local access can retrieve sensitive data from this cache even when users are not authenticated.

For instance in a long running Single Page Application (SPA), one user may "log out", but then another user may access the application without refreshing, in which case all the cached data is still available.

26.Differentiate named routing n custom directives...?

21st & 26th same

61.how to create dynamic buttons in angular js

Requirements is to add html content dynamically and that content might have a click event on it.

So the code Angular code I have below displays a button, and when clicked, it dynamically adds another button. Clicking on the dynamically added buttons, should add another button, but I cannot get the `ng-click` to work on the dynamically added buttons

Index.html

```

<!DOCTYPE html>
<html lang="en" >
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js">
</script>
<script src="app.js"></script>
<link rel="stylesheet" href="my_style.css">
<script src="controllers/ctrl.js"></script>
<div ng-app="app" ng-controller="Ctrl">
  <button class="addfields" ng-click="addNewChoice()">Add choice</button>
  <fieldset data-ng-repeat="choice in choicesA">
    <input type="text" ng-model="choice.name" name="" placeholder="Enter name">
    <button class="remove" ng-click="removeChoice($index)">-</button>
    <button class="addfields" ng-click="addNewChoiceB(choice)">Add fields</button>
    <div data-ng-repeat="choiceb in choice.choicesB">
      <input type="text" ng-model="choiceb.name" name="" placeholder="Enter field">
      <button class="remove" ng-click="removeChoiceB(choice,$index)">-</button>
    </div>
  </fieldset>
  <div id="choicesDisplay">
    <pre>choicesA = {{ choicesA }}</pre>
    <pre data-ng-repeat="choiceb in choicesA">choicesB = {{ choiceb.choicesB }}</pre>
  </div>
</div>
</html>

```

app.js

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

```
my_style.css
```

```

fieldset {
  background: #FCFCFC;
  padding: 16px;
  border: 1px solid #D5D5D5;
}
.addfields {
  margin: 10px 0;
}
#choicesDisplay {
  padding: 10px;
  background: rgb(227, 250, 227);
  border: 1px solid rgb(171, 239, 171);
}

```

```
    color: rgb(9, 56, 9);
}
.remove {
    background: #C76868;
    color: #FFF;
    font-weight: bold;
    font-size: 21px;
    border: 0;
    cursor: pointer;
    display: inline-block;
    padding: 4px 9px;
    vertical-align: top;
    line-height: 100%;
}
input[type="text"],
select {
    padding: 5px;
}
```

Ctrl.js

```
app.controller('Ctrl',Ctrl);
Ctrl.$inject("$scope")
function Ctrl($scope) {
    $scope.choicesA = [{
        id: 'choice1',
        choicesB:[]
    }, {
        id: 'choice2',
        choicesB:[]
    }];
    $scope.addNewChoice = function() {
        var newItemNo = $scope.choicesA.length + 1;
        $scope.choicesA.push({
            'id': 'choice' + newItemNo,
            choicesB:[]
        });
    };
    $scope.removeChoice = function(ind) {
        $scope.choicesA.splice(ind,1);
    };
}
```

```
$scope.addNewChoiceB = function(choice) {  
    var newItemNo = choice.choicesB.length + 1;  
    choice.choicesB.push({  
        'id': 'choice' + newItemNo  
    });  
};  
  
$scope.removeChoiceB = function(choice,ind) {  
    choice.choicesB.splice(ind,1);  
};  
};
```

62.what are the types of scopes available in custom directives

Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

This is to ease the understanding of the \$scope in the custom directives. While creating custom directives we have to apply different types of functionalities, so for that we must know about the \$scope behavior in the directives.

\$scope has a very important role in AngularJS, it works as a mediator (or like a glue) between the logic & view in an Angular application. Now if we talk about the custom directives, then first question which arises is-

“In any application if we want some specific functionality and we want to reuse that in whole application module, then for this we need to develop a set of code. Angular calls it directives.”

I am not going to discuss so much about custom directives basics here. In this blog I am just focusing on use of \$scope into directives.

So, when we create a custom directive it has a default scope, which is the parent scope (the controller's scope from where the directive is called). This behavior is by default, until and unless we do not set the scope.

As we know that whenever we define a directive, there is a “directive definition object” (DDO), in which we set some parameters like- restrict, template, require, scope etc.

In this blog I will talk about the scope properties, they are false, true, {}.

Let's discuss them one by one.

Scope : false (Shared Scope)

In layman language false is assumed as no, so if the scope is set to false, then it means use parent scope in the directive and do not create a new scope for the directive itself. It just uses the scope of respective controller. So let us suppose if we have a controller named “homeController” and a directive “printName”, then in printName directive we will get parent scope by default and any change in scope values, either child or parent, will reflect in both.

Example:*Code Snippet*

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

app.controller("sharedController", function ($scope) {
    $scope.name = "rock";
});

app.directive("sharedDirective", function () {
    return {
        restrict: "EA",
        scope: false,
        template: "<div>directive scope value : {{name}}</div>" +
            "Change directive scope value : <input type='text' ng-model='name' />"
    };
});

//view (html)
<div ng-app="blogDemo">
    <div ng-controller="sharedController">
        <h2>parent scope value {{name}} </h2>
        <div shared-directive ></div>
    </div>
</div>
```

In this example, we can see whenever the value of parent scope changes, it will reflect in the directive scope also, because they both are sharing the same scope object.

Scope : true (Inherited Scope)

Using this property, the directive will create a new scope for itself. And inherit it from parent scope. If we do any changes to the controller scope it will reflect on directive scope, but it won't work the other way around. This is because both of them use their own copies of scope object.

Example:*Code Snippet*

```
//module, controller, directive
var app = angular.module("blogDemo", []);

app.controller("inheritedController", function ($scope) {
    $scope.orgName = "Quovantis Parent";
});

app.directive("inheritedDirective", function () {
    return {
        restrict: "EA",
        scope: true,
```

```
template: "<div>my organisation name is : {{orgName}}</div> type for change name :  
<input type='text' ng-model='orgName' />"  
};  
});  
//view (html)  
<div ng-app="blogDemo">  
  <div ng-controller="inheritedController">  
    <h2>parent scope value {{orgName}} </h2>  
    <div inherited-directive ></div>  
  </div>  
</div>
```

In this example when the first time directive loads, the screen will show the value of parent scope. But when we will change the value from text box. Then this will only change into child scope only. Means no change in parent scope.

Scope : {} (Isolated Scope)

One of the important features, its called isolated scope. Here too the directive will create a new scope object but it is not inherited by the parent scope, so now this scope doesn't know anything about the parent scope.

But the question arises, if we do not have the link from parent scope then how can we get the values from it, and how can we modify it ?

The answer is- set the objects property into DDO, but for this it is necessary to set on attributes into the directive.

In isolated scope we use three prefixes which helps to bind the property or methods from the controller (parent scope) to directive (isolated scope). Lets understand how this works. Whenever a directive finds any prefixes in its scope property in DDO, it checks it in directive declaration (in html page where the directive is called) with attribute declared on this element. We can also change the name by giving a separate attribute name after any of the prefixes.

These are @, =, &

'@' : One way binding

One way binding means a parent sending anything to the directive scope through the attribute, gets reflected in the directive. But if any change in the directive happens it will not reflect in parent. The @ is used to pass string values.

Example:

Code Snippet

```
//module  
var app = angular.module('quovantisBlog', []);  
//controller
```



```
app.controller('OneWayController', ['$scope', function ($scope) {
    $scope.student = {
        name: 'Rohit',
        class: 'MCA',
        Address: 'New Delhi'
    };
}]);
// directive
app.directive('oneWayDirective', function () {
    return {
        scope: {
            name: '@'
        },
        template: 'Student Name: {{ name }}'
    };
});
//view (html)
<one-way-directive name="{{ student.name }}"></one-way-directive>
or
<one-way-directive studName="{{ student.name }}"></one-way-directive>
then directive would be with a change in scope property.
```

Code Snippet

```
app.directive('oneWayDirective', function () {
    return {
        scope: {
            name: '@studName'
        },
        template: 'Student Name: {{ name }}'
    };
});
```

'=' : Two way binding

This is called two way binding, because the parent scope will also reflect to directive scope vice-versa.

It is used for passing object to the directive instead of string. This object could be changed from both sides, from parent or from directive. That is why it is called two-way.

Example:*Code Snippet*

```
//module
var blogDemo = angular.module('myApp',[]);

//directive
blogDemo.directive('twoWayDirective', function() {
  return {
    restrict: 'EA',
    scope: { obj: "=" },
    template: '<div>Welcome, {{obj.fname + obj.lname}}!</div>'
  };
});

//controller
blogDemo.controller('blogController', function ($scope) {
  $scope.obj = { fname: "shubh", lname: "raj" };
});

//view (html)
<div ng-controller="blogController">
  <two-way-directive obj="obj"></two-way-directive>
</div>
```

'&' : Method binding

Used to bind any parent's method to directive scope. Whenever we want to call the parent methods from the directive we can use this. It is used to call external (outside of current scope) functions. Overall "&" is used to pass data as a function or method.

Example:*Code Snippet*

```
//module
var blogDemo = angular.module('myApp',[]);

//directive
blogDemo.directive('methodDirective', function() {
  return {
    scope: {
      studData: '=',
      swap: '&'
    },
    template: '<div>the changed names are, {{obj.fname + obj.lname}}!</div>'+
      '<button id="btn1" ng-click="swap()">Click here to Swap student Data</button>'
  };
});
```

```
});  
//controller  
blogDemo.controller('blogController', function ($scope) {  
    $scope.studData = { fname: "shubh", lname: "raj" };  
    $scope.swapData = function () {  
        $scope.customer = {  
            fname: 'Raj',  
            lname: 'kumar'  
        };  
    };  
});  
//view (html)  
<div ng-controller="blogController">  
    <method-directive studData="studData" swap="swapData()"></method-directive>  
</div>
```

In this example the directive creates a property inside its local scope, that is swapData. We can also understand swap as an alias for swapData. So we pass a method to '&' which is then invoked by the directive whenever required.

Summarizing, Shared scope (sharing the same scope and data, can not pass the data explicitly), Inherited scope (parent scope values can be fetched into child but child means directive scope will not effect parent), Isolated scope (both controller & directive do not share the scope & data, we can explicitly pass the data using some parameters that is @, &, =).

63.What is component? How do we create it?

In AngularJS, a Component is a special kind of directive that uses a simpler configuration which is suitable for a component-based application structure.

This makes it easier to write an app in a way that's similar to using Web Components or using the new Angular's style of application architecture.

Advantages of Components:

- simpler configuration than plain directives

- promote sane defaults and best practices

- optimized for component-based architecture

- writing component directives will make it easier to upgrade to Angular

When not to use Components:

- for directives that need to perform actions in compile and pre-link functions, because they aren't available

- when you need advanced directive definition options like priority, terminal, multi-element

- when you want a directive that is triggered by an attribute or CSS class, rather than an

element

Creating and configuring a component

Components can be registered using the `.component()` method of an AngularJS module (returned by `angular.module()`). The method takes two arguments:

The name of the Component (as string).The Component config object. (Note that, unlike the `.directive()` method, this method does not take a factory function.)

Index.js

```
(function(angular) {  
  'use strict';  
  angular.module('heroApp', []).controller('MainCtrl', function MainCtrl() {  
    this.hero = {  
      name: 'Spawn'  
    };  
  });  
})(window.angular);
```

heroDetail.js

```
(function(angular) {  
  'use strict';  
  function HeroDetailController() {  
  }  
  angular.module('heroApp').component('heroDetail', {  
    templateUrl: 'heroDetail.html',  
    controller: HeroDetailController,  
    bindings: {  
      hero: '='  
    }  
  });  
})(window.angular);
```

Index.html

```
<!doctype html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>Example - example-heroComponentSimple-production</title>  
  <script src=  
    "https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"></script>  
  <script src="index.js"></script>  
  <script src="heroDetail.js"></script>  
</head>
```

```
<body ng-app="heroApp">
  <!-- components match only elements -->
  <div ng-controller="MainCtrl as ctrl">
    <b>Hero</b><br>
    <hero-detail hero="ctrl.hero"></hero-detail>
  </div>
</body>
</html>
```

heroDetail.html

```
<span>Name: {{$ctrl.hero.name}}</span>
```

64.What are core features of Angular JS

AngularJS is a great JavaScript framework that has some very compelling features for not only developers, but designers as well! In this tutorial, we will cover what I consider to be the most essential features, and how they can help make your next web application awesome.

To get an idea of what you can do with AngularJS, check out the range of AngularJS items on Envato Market. You can find an image cropper, an eCommerce web application, a JSON editor, and much more.

The Elevator Pitch: AngularJS in a Nutshell

AngularJS is a new, powerful, client-side technology that provides a way of accomplishing really powerful things in a way that embraces and extends HTML, CSS and JavaScript, while shoring up some of its glaring deficiencies. It is what HTML would have been, had it been built for dynamic content.

In this article, we will cover a few of the most important AngularJS concepts to get the "big picture." It is my goal that, after seeing some of these features, you will be excited enough to go and build something fun with AngularJS.

Feature 1: Two Way Data-Binding

Think of your model as the single-source-of-truth for your application. Your model is where you go to to read or update anything in your application.

Data-binding is probably the coolest and most useful feature in AngularJS. It will save you from writing a considerable amount of boilerplate code. A typical web application may contain up to 80% of its code base, dedicated to traversing, manipulating, and listening to the DOM. Data-binding makes this code disappear, so you can focus on your application.

Think of your model as the single-source-of-truth for your application. Your model is where you go to to read or update anything in your application. The data-binding directives provide a projection of your model to the application view. This projection is seamless, and occurs without any effort from you.

Traditionally, when the model changes, the developer is responsible for manually manipulating the DOM elements and attributes to reflect these changes. This is a two-way street. In one direction, the model changes drive change in DOM elements. In the other, DOM element changes necessitate changes in the model. This is further complicated by user interaction, since the developer is then responsible for interpreting the interactions, merging them into a model, and updating the view. This is a very manual and cumbersome process, which becomes difficult to control, as an application grows in size and complexity. There must be a better way! AngularJS' two-way data-binding handles the synchronization between the DOM and the model, and vice versa.

Here is a simple example, which demonstrates how to bind an input value to an <h1> element.

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/angular-1.0.0rc10.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <hr>
      <h1>Hello, {{yourName}}!</h1>
    </div>
  </body>
</html>
```

This is extremely simple to set up, and almost magical...

Feature 2: Templates

It's important to realize that at no point does AngularJS manipulate the template as strings. It's all the browser DOM.

In AngularJS, a template is just plain-old-HTML. The HTML vocabulary is extended, to contain instructions on how the model should be projected into the view.

The HTML templates are parsed by the browser into the DOM. The DOM then becomes the input to the AngularJS compiler. AngularJS traverses the DOM template for rendering instructions, which are called directives. Collectively, the directives are responsible for setting up the data-binding for your application view.

It is important to realize that at no point does AngularJS manipulate the template as strings. The input to AngularJS is browser DOM and not an HTML string. The data-bindings are DOM transformations, not string concatenations or innerHTML changes. Using the DOM as the input, rather than strings, is the biggest differentiation AngularJS has from its sibling

frameworks. Using the DOM is what allows you to extend the directive vocabulary and build your own directives, or even abstract them into reusable components!

One of the greatest advantages to this approach is that it creates a tight workflow between designers and developers. Designers can mark up their HTML as they normally would, and then developers take the baton and hook in functionality, via bindings with very little effort. Here is an example where I am using the `ng-repeat` directive to loop over the images array and populate what is essentially an `img` template.

```
function AlbumCtrl($scope) {  
    scope.images = [  
        {"thumbnail": "img/image_01.png", "description": "Image 01 description"},  
        {"thumbnail": "img/image_02.png", "description": "Image 02 description"},  
        {"thumbnail": "img/image_03.png", "description": "Image 03 description"},  
        {"thumbnail": "img/image_04.png", "description": "Image 04 description"},  
        {"thumbnail": "img/image_05.png", "description": "Image 05 description"}  
    ];  
}  
  
<div ng-controller="AlbumCtrl">  
    <ul>  
        <li ng-repeat="image in images">  
              
        </li>  
    </ul>  
</div>
```

It is also worth mentioning, as a side note, that AngularJS does not force you to learn a new syntax or extract your templates from your application.

Feature 3: MVC

AngularJS incorporates the basic principles behind the original MVC software design pattern into how it builds client-side web applications.

The MVC or Model-View-Controller pattern means a lot of different things to different people. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel).

The Model:

The *model* is simply the data in the application. The *model* is just plain old JavaScript objects. There is no need to inherit from framework classes, wrap it in proxy objects, or use special getter/setter methods to access it. The fact that we are dealing with vanilla JavaScript is a really nice feature, which cuts down on the application boilerplate.

The ViewModel:

A *viewmodel* is an object that provides specific data and methods to maintain specific views. The *viewmodel* is the *\$scope* object that lives within the AngularJS application. *\$scope* is just a simple JavaScript object with a small API designed to detect and broadcast changes to its state.

The Controller:

The *controller* is responsible for setting initial state and augmenting *\$scope* with methods to control behavior. It is worth noting that the *controller* does not store state and does not interact with remote services.

The View:

The *view* is the HTML that exists after AngularJS has parsed and compiled the HTML to include rendered markup and bindings.

This division creates a solid foundation to architect your application. The *\$scope* has a reference to the data, the *controller* defines behavior, and the *view* handles the layout and handing off interaction to the *controller* to respond accordingly.

Feature 4: Dependency Injection

AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

Dependency Injection (DI) allows you to ask for your dependencies, rather than having to go look for them or make them yourself. Think of it as a way of saying "Hey I need X", and the DI is responsible for creating and providing it for you.

To gain access to core AngularJS services, it is simply a matter of adding that service as a parameter; AngularJS will detect that you need that service and provide an instance for you.

```
function EditCtrl($scope, $location, $routeParams) {  
    // Something clever here...  
}
```

You are also able to define your own custom services and make those available for injection as well.

```
angular.  
    module('MyServiceModule', []).  
    factory('notify', ['$window', function (win) {  
        return function (msg) {  
            win.alert(msg);  
        };  
    }]);
```

```
function myController(scope, notifyService) {  
    scope.callNotify = function (msg) {  
        notifyService(msg);  
    };  
}
```



```
myController.$inject = ['$scope', 'notify'];
```

Feature 5: Directives

Directives are my personal favorite feature of AngularJS. Have you ever wished that your browser would do new tricks for you? Well, now it can! This is one of my favorite parts of AngularJS. It is also probably the most challenging aspect of AngularJS.

Directives can be used to create custom HTML tags that serve as new, custom widgets. They can also be used to "decorate" elements with behavior and manipulate DOM attributes in interesting ways.

Here is a simple example of a directive that listens for an event and updates its \$scope, accordingly.

```
myModule.directive('myComponent', function(mySharedService) {
  return {
    restrict: 'E',
    controller: function($scope, $attrs, mySharedService) {
      $scope.$on('handleBroadcast', function() {
        $scope.message = 'Directive: ' + mySharedService.message;
      });
    },
    replace: true,
    template: '<input>'
  };
});
```

Then, you can use this custom directive, like so.

```
1 <my-component ng-model="message"></my-component>
```

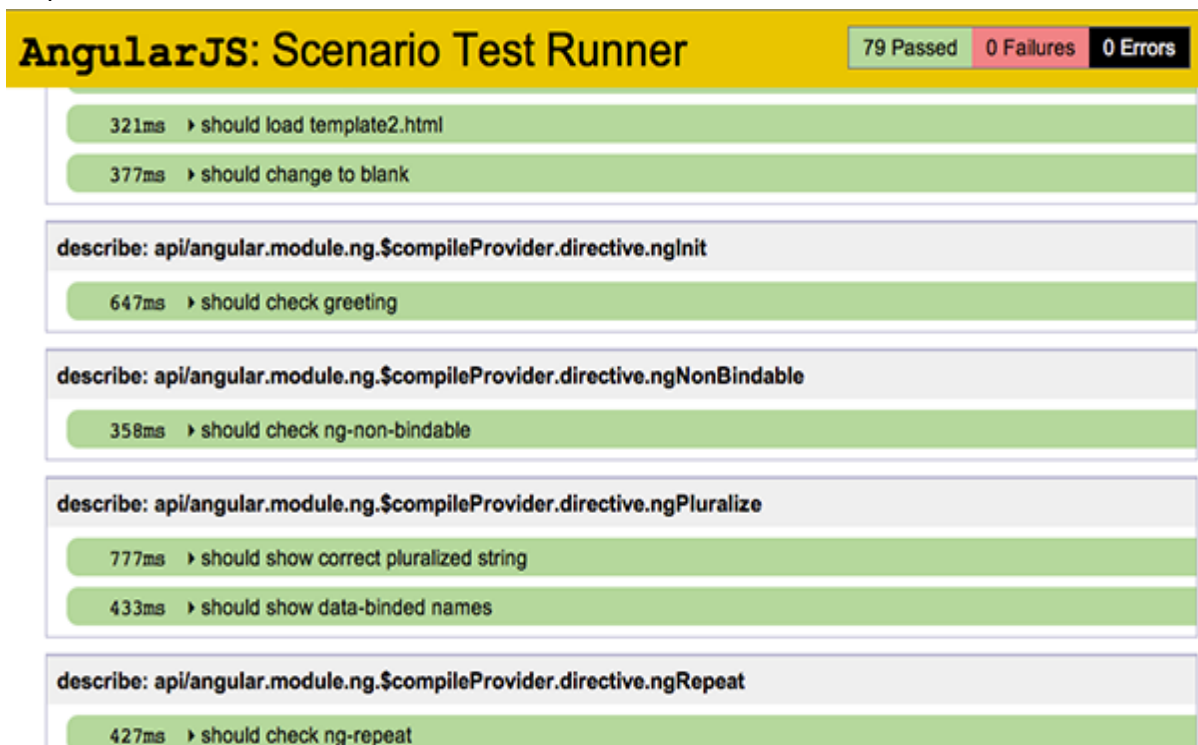
Creating your application as a composition of discrete components makes it incredibly easy to add, update or delete functionality as needed.

Bonus Feature: Testing

The AngularJS team feels very strongly that any code written in JavaScript needs to come with a strong set of tests. They have designed AngularJS with testability in mind, so that it makes testing your AngularJS applications as easy as possible. So there's no excuse for not doing it.

Given the fact that JavaScript is dynamic and interpreted, rather than compiled, it is extremely important for developers to adopt a disciplined mindset for writing tests. AngularJS is written entirely from the ground up to be testable. It even comes with an end-to-end and unit test runner setup. If you would like to see this in action, go check out the angular-seed project at <https://github.com/angular/angular-seed>.

Once you have the seed project, it's a cinch to run the tests against it. Here is what the output looks like:



AngularJS: Scenario Test Runner 79 Passed 0 Failures 0 Errors

- 321ms ▶ should load template2.html
- 377ms ▶ should change to blank
- describe: api/angular.module.ng.\$compileProvider.directive.ngInit
 - 647ms ▶ should check greeting
- describe: api/angular.module.ng.\$compileProvider.directive.ngNonBindable
 - 358ms ▶ should check ng-non-bindable
- describe: api/angular.module.ng.\$compileProvider.directive.ngPluralize
 - 777ms ▶ should show correct pluralized string
 - 433ms ▶ should show data-binded names
- describe: api/angular.module.ng.\$compileProvider.directive.ngRepeat
 - 427ms ▶ should check ng-repeat

The API documentation is full of end-to-end tests that do an incredible job of illustrating how a certain part of the framework should work. After a while, I found myself going straight to the tests to see how something worked, and then maybe reading the rest of the documentation to figure something out.

65.What is the exact use of \$inject?

That is one approach to support Dependency Injection after your code is minified (if you choose to minify).

When you declare a controller, the function takes parameters:

```
function ($scope,notify)
```

When you minify the code, your function will look like this:

```
function (a,b)
```

Since AngularJS uses the function parameter names to infer DI, your code will break because AngularJS doesn't know about a or b.

To solve this problem, they provided additional ways to declare controllers (or other services/factories/etc) for that matter:

1) For controllers, use the `$inject` method - here you pass an array of literals that map to the parameters of your controller function. So, if you provide

```
["$scope","notify"]
```

then the value of the first parameter to your function will be the a scope object associated with this controller and the second parameter will be the notify service.

2) When declaring new controllers, services, etc, you can use the array literal syntax. Here, you do something like this:

```
angular.module("myModule").controller("MyController",["$scope","notify",function($scope,notify){
.....
}]);
```

The array as a parameter to the controller function maps the DI objects to your function parameters

96.Explain directives ng-if, ng-switch and ng-show.

ngif

==>The ng-if directive removes the HTML element if the expression evaluates to false.

==>If the if statement evaluates to true, a copy of the Element is added in the DOM.

==>The ng-if directive is different from the ng-hide, which hides the display of the element, where the ng-if directive completely removes the element from the DOM.

==>The ng-if directive will destroy (completely removes) the element or recreate the element.

==>Supported by all HTML elements.

==>This directive can add / remove HTML elements from the DOM based on an expression.

==>If the expression is true, it add HTML elements to DOM, otherwise HTML elements are removed from the DOM.

Syntax:

==>ng-if, on the other hand, actually removes the element from the DOM

when the condition is false and only adds the element back once the condition turns true.

```
<element ng-if="expression"></element>
```

Parameter Values:

Value	Description
-------	-------------

expression	An expression that will completely remove the element if it returns false. If it returns true, a copy of the element will be inserted instead.
------------	---

EXAMPLE:

```
<!DOCTYPE html>
<html>
<body ng-app="">
Keep HTML: <input type="checkbox" ng-model="myVar" ng-init="myVar = true">
<div ng-if="myVar">
<h1>Welcome</h1>
<p>Welcome to my if directive</p>
<hr>
</div>
<p>The DIV element will be removed when the checkbox is not checked.</p>
<p>The DIV element will return, if you check the checkbox.</p>
</body>
</html>
```

97.What is ng-repeat directive?**ng-repeat:**

==>The ng-repeat directive repeats a set of HTML, a given number of times.

==>The set of HTML will be repeated once per item in a collection.

==>The collection must be an array or an object.

Note: Each instance of the repetition is given its own scope, which consist of the current item.

==>If you have an collection of objects, the ng-repeat directive is perfect for making a HTML table,

displaying one table row for each object, and one table data for each object property.

Syntax:

```
<element ng-repeat="expression"></element>
```

Supported by all HTML elements.

Parameter Values

Value	Description
-------	-------------

expression An expression that specifies how to loop the collection.

Legal Expression examples:

x in records

(key, value) in myObj

x in records track by \$id(x)

Example:

<!DOCTYPE html>
<html>
<body ng-app="myApp" ng-controller="myCtrl">
<h1 ng-repeat="x in records">{{x}}</h1>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function(\$scope) {
 \$scope.records = [
 "Sachin Tendulkar",
 "A.P.J Abdul Kalam",
 "Abraham Lincoln",
 "Nelson Mandela",
]
});
</script>
</body>
</html>

98.Explain ng-include directive?

AngularJS ng-include Directive

=====
==>The ng-include directive is useful if we want to include an external resource in an HTML template.

Definition and Usage:

=====
==>The ng-include directive includes HTML from an external file.

==>The included content will be included as childnodes of the specified element.

==>The value of the ng-include attribute can also be an expression, returning a filename.

==>By default, the included file must be located on the same domain as the document.

==>Supported by all HTML elements.

Syntax:

```
<element ng-include="filename" onload="expression" autoscroll="expression" ></element>
```

(or)

==>The ng-include directive can also be used as an element:

```
<ng-include src="filename" onload="expression" autoscroll="expression" ></ng-include>
```

Parameter Values

Value	Description
-------	-------------

filename	A filename, written with apostrophes, or an expression which returns a filename.
onload	Optional. An expression to evaluate when the included file is loaded.
autoscroll	Optional. Whether or not the included section should be able to scroll into a specific view.

AngularJS - Includes

==>HTML does not support embedding html pages within html page.

==> To achieve this functionality following ways are used –

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 a HTML page using ng-include directive.

```
<div ng-app = "" ng-controller = "studentController">
  <div ng-include = ""main.html""></div>
  <div ng-include = ""subjects.html""></div>
</div>
```

==>By default, HTML does not provide the facility to include client side code from other files.

==>

Client Side includes

Server Side Includes

AngularJS Includes

Client Side includes:

One of the most common ways is to include HTML code is via Javascript.

JavaScript is a programming language which can be used to manipulate the content

in an HTML page on the fly. Hence, Javascript can also be used to include code from other files.

Server Side Includes:

Server Side Includes are also available for including a common piece of code throughout a site. This is normally done for including content in the below parts of an HTML document.

Page header

Page footer

Navigation menu.

Note:

The virtual parameter is used to specify the file (HTML page, text file, script, etc.) that needs to be included.

If the web server process does not have access to read the file or execute the script, the include command will fail. The 'virtual' word is a keyword that is required to be placed in the include directive.

AngularJS Includes:

==>Angular provides the function to include the functionality from other AngularJS files by using the ng-include directive.

==>The primary purpose of the "ng-include directive" is used to fetch, compile and include an external HTML fragment in the main AngularJS application.

Summary:

==>By default, we know that HTML does not provide a direct way to include HTML content from other files like other programming languages.

==>Javascript along with JQuery is the best-preferred option for embedding HTML content from other files.

==>Another way of including HTML content from other files is to use the <include> directive and the virtual parameter keyword.

==>The virtual parameter keyword is used to denote the file which needs to be embedded. This is known as server-side includes.

==>Angular also provides the facility to include files by using the ng-include directive.

This directive can be used to inject code from external files directly into the main HTML file.

Example:

```
<!DOCTYPE html>
<html>
<body ng-app="">
<div ng-include="include.html"></div>
</body>
</html>
```

include.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Include Header..!!</h1><br><br>
<h5>This text has been included into the HTML page, using ng-include!</h5>
</body>
</html>
```

99.What is template in angularjs?**Template:**

==>In AngularJS, templates are written with HTML that contains AngularJS-specific elements and attributes.

==>AngularJS combines the template with information from the model and controller to render the dynamic view that a user sees in the browser.

==>In a simple app, the template consists of HTML, CSS, and AngularJS directives contained in just one HTML file (usually index.html).

==>In a more complex app, you can display multiple views within one main page using "partials" – segments of template located in separate HTML files.

You can use the ngView directive to load partials based on configuration passed to the \$route service.

These are the types of AngularJS 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 AngularJS markup.

--Filter — Formats data for display.

--Form controls — Validates user input.

What are Website Templates?

==>A website template (or web template) is a pre-designed webpage, or set of HTML webpages

that anyone can use to "plug-in" their own text content and images into to create a website.

==> Usually built with HTML and CSS code, website templates allow anyone to setup a website

without having to hire a professional web developer or designer, although, many developers do use website templates to create sites for their clients.

NareshIT

Types of Web Templates:

==>Webpage templates may be self contained zip file downloads, or part of a proprietary web builder interface, or included with an html software editing program.

==>They may be responsive, adaptive or static in design or configured specifically for mobile.

==>The file extension can be .html .htm .php or .asp. In all cases they will be built using html and css code.

==>If it's labeled as "responsive", the layout will flex in width to fit the specific device viewing screen regardless of whether it's a desktop computer, tablet or smartphone.

Q)What Can a Web Template Include?

Text and .jpg, .png or .gif images, jQuery and CSS3 animation, shopping carts, contact forms, dynamic image galleries and slideshows, a PDF download links page, and video players (including embedded Youtube movies) are just a few of the features that can be built into a design.

==>Web template designs and code vary widely from vendor to vendor, so when selecting a website template, you will want to make sure it already includes the functions, scripts and applications you require for your web development project.

==>Text, stock photos, scripts and 3rd party plugins can be added to the ready-made pages.

==>The included stock images can be replaced with the users own .jpg images or edited as the project requires.

Mobile and/or Responsive:

==>Web templates come in many flavors so you should carefully consider your options before

committing to a design to use. It's best to choose one that will be compliant and viewable on mobile and also passes the Google Mobile-Friendly Test. To read about the pros and cons of

these options see choosing responsive or mobile.

Types of Mobile Web Templates:

--Responsive

--Full website with mobile version (adaptive)

--Mobile & mobile optimized

--Mobile upgrade (for older websites)

Initially it was developed by MisKo Hevery and

Adam Abrons. Currently it is being developed by

Google.

100. Who created Angular JS ?

The History of AngularJS

==>AngularJS was created, as a side project, in 2009 by two developers, Misko Hevery and Adam Abrons.

==>The two originally envisioned their project, GetAngular, to be an end-to-end tool that allowed web designers to interact with both the frontend and the backend.

==>Hevery eventually began working on a project at Google called Google Feedback.

==>Hevery and 2 other developers wrote 17,000 lines of code over the period of 6 months for Google Feedback. However, as the code size increased, Hevery began to grow frustrated with how difficult it was to test and modify the code the team had written.

==>So Hevery made the bet with his manager that he could rewrite the entire application using his side GetAngular project in two weeks. Hevery lost the bet. Instead of 2 weeks it took him 3 weeks to rewrite the entire application, but he was able to cut the application from 17,000 lines to 1,500 lines. (For a talk from Misko Hevery about the full history up until January 2014, see his keynote from ngConf 2014).

==>Because of Hevery's success, his manager took notice, and Angular.js development began to accelerate from there.

Q) Who is Behind AngularJS?

Google.

==>One of the original creators, Adam Abrons stopped working on AngularJS but Misko Hevery

and his manager, Brad Green, spun the original GetAngular project into a new project, named it AngularJS, and built a team to create and maintain it within Google.

==>One of AngularJS' first big wins internally at Google occurred when the company DoubleClick

was acquired by Google and they started rewriting part of their application using AngularJS. Because of DoubleClick's initial success, Google seems to have invested even more resources

into Angular and has blessed AngularJS for internal and external product usage.

Because of this, the Angular team inside Google has grown rapidly.

Inline Editor

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8"/>
    <title>Learn AngularJS - Inline Editor</title>

    <link href="http://fonts.googleapis.com/css?family=Open+Sans:400,700"
rel="stylesheet" />

    <!-- The main CSS file -->
    <link href="style.css" rel="stylesheet" />

    <!--[if lt IE 9]>
      <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
  </head>

  <!-- Notice the controller directive -->
  <body ng-app ng-controller="InlineEditorController">

    <!-- When this element is clicked, hide the tooltip -->
    <div id="main" ng-click="hideTooltip()">

      <!-- This is the tooltip. It is shown only when the showtooltip variable
is truthful -->
      <div class="tooltip" ng-click="$event.stopPropagation()" ng-
show="showtooltip">

        <!-- ng-model binds the contents of the text field with the
"value" model.

        Any changes to the text field will automatically update
the value, and

        all other bindings on the page that depend on it. -->

        <input type="text" ng-model="value" />
      </div>
```

```
        <!-- Call a method defined in the InlineEditorController that toggles
        the showtooltip variable -->
        <p ng-click="toggleTooltip($event)">{{value}}</p>

    </div>

    <!-- Include AngularJS from Google's CDN -->
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
    <script src="script.js"></script>
    </body>
</html>
```

```
// The controller is a regular JavaScript function. It is called
// once when AngularJS runs into the ng-controller declaration.
```

```
function InlineEditorController($scope){

    // $scope is a special object that makes
    // its properties available to the view as
    // variables. Here we set some default values:

    $scope.showtooltip = false;
    $scope.value = 'Edit me.';

    // Some helper functions that will be
    // available in the angular declarations

    $scope.hideTooltip = function(){

        // When a model is changed, the view will be automatically
        // updated by by AngularJS. In this case it will hide the tooltip.

        $scope.showtooltip = false;
    }

    $scope.toggleTooltip = function(e){
        e.stopPropagation();
        $scope.showtooltip = !$scope.showtooltip;
    }
}
```

```
}
```

```
/*-----  
    Simple reset  
-----*/
```

```
*{  
    margin:0;  
    padding:0;  
}
```

```
/*-----  
    General Styles  
-----*/
```

```
body{  
    font:15px/1.3 'Open Sans', sans-serif;  
    color: #5e5b64;  
    text-align:center;  
}
```

```
a, a:visited {  
    outline:none;  
    color:#389dc1;  
}
```

```
a:hover{  
    text-decoration:none;  
}
```

```
section, footer, header, aside, nav{  
    display: block;  
}
```

```
/*-----  
    The edit tooltip
```

-----*/

```
.tooltip{
  background-color:#5c9bb7;

  background-image:-webkit-linear-gradient(top, #5c9bb7, #5392ad);
  background-image:-moz-linear-gradient(top, #5c9bb7, #5392ad);
  background-image:linear-gradient(top, #5c9bb7, #5392ad);

  box-shadow: 0 1px 1px #ccc;
  border-radius:3px;
  width: 290px;
  padding: 10px;

  position: absolute;
  left:50%;
  margin-left:-150px;
  top: 80px;
}

.tooltip:after{
  content:"";
  position:absolute;
  border:6px solid #5190ac;
  border-color:#5190ac transparent transparent;
  width:0;
  height:0;
  bottom:-12px;
  left:50%;
  margin-left:-6px;
}

.tooltip input{
  border: none;
  width: 100%;
  line-height: 34px;
  border-radius: 3px;
  box-shadow: 0 2px 6px #bbb inset;
  text-align: center;
  font-size: 16px;
```

```
    font-family: inherit;
    color: #8d9395;
    font-weight: bold;
    outline: none;
}

p{
    font-size:22px;
    font-weight:bold;
    color:#6d8088;
    height: 30px;
    cursor:default;
}

p b{
    color:#ffffff;
    display:inline-block;
    padding:5px 10px;
    background-color:#c4d7e0;
    border-radius:2px;
    text-transform:uppercase;
    font-size:18px;
}

p:before{
    content:'📧';
    display:inline-block;
    margin-right:5px;
    font-weight:normal;
    vertical-align: text-bottom;
}

#main{
    height:300px;
    position:relative;
    padding-top: 150px;
}
```


Instant Search

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8"/>
    <title>Learn AngularJS - Instant Search</title>

    <link
href="http://fonts.googleapis.com/css?family=Cookie|Open+Sans:400,700" rel="stylesheet"
/>

    <!-- The main CSS file -->
    <link href="style.css" rel="stylesheet" />

    <!--[if lt IE 9]>
      <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
  </head>

  <!-- Initialize a new AngularJS app and associate it with a module named
"instantSearch"-->
  <body ng-app="instantSearch" ng-controller="InstantSearchController">

    <div class="bar">
      <!-- Create a binding between the searchString model and the text
field -->
      <input type="text" ng-model="searchString" placeholder="Enter your
search terms" />
    </div>

    <ul>
      <!-- Render a li element for every entry in the items array. Notice
the custom search filter "searchFor". It takes the value of the
searchString model as an argument.
-->
      <li ng-repeat="i in items | searchFor:searchString">
        <a href="{{i.url}}"></a>
```

```
        <p>{{i.title}}</p>
    </li>
</ul>

    <!-- Include AngularJS from Google's CDN -->
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

```
// Define a new module for our app
var app = angular.module("instantSearch", []);

// Create the instant search filter

app.filter('searchFor', function(){

    // All filters must return a function. The first parameter
    // is the data that is to be filtered, and the second is an
    // argument that may be passed with a colon (searchFor:searchString)

    return function(arr, searchString){

        if(!searchString){
            return arr;
        }

        var result = [];

        searchString = searchString.toLowerCase();

        // Using the forEach helper method to loop through the array
        angular.forEach(arr, function(item){

            if(item.title.toLowerCase().indexOf(searchString) !== -1){
                result.push(item);
            }

        });

    });
});
```

```
        return result;
    };

});

// The controller

function InstantSearchController($scope){

    // The data model. These items would normally be requested via AJAX,
    // but are hardcoded here for simplicity. See the next example for
    // tips on using AJAX.

    $scope.items = [
        {
            url: 'http://tutorialzine.com/2013/07/50-must-have-plugins-for-
extending-twitter-bootstrap/',
            title: '50 Must-have plugins for extending Twitter Bootstrap',
            image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/07/featured_4-100x100.jpg'
        },
        {
            url: 'http://tutorialzine.com/2013/08/simple-registration-system-php-
mysql/',
            title: 'Making a Super Simple Registration System With PHP and
MySQL',
            image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/08/simple_registration_system-100x100.jpg'
        },
        {
            url: 'http://tutorialzine.com/2013/08/slideout-footer-css/',
            title: 'Create a slide-out footer with this neat z-index trick',
            image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/08/slide-out-footer-100x100.jpg'
        },
        {
            url: 'http://tutorialzine.com/2013/06/digital-clock/',
            title: 'How to Make a Digital Clock with jQuery and CSS3',
            image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/06/digital_clock-100x100.jpg'
        },
    ],
```

```
{
    url: 'http://tutorialzine.com/2013/05/diagonal-fade-gallery/',
    title: 'Smooth Diagonal Fade Gallery with CSS3 Transitions',
    image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/05/featured-100x100.jpg'
},
{
    url: 'http://tutorialzine.com/2013/05/mini-ajax-file-upload-form/',
    title: 'Mini AJAX File Upload Form',
    image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/05/ajax-file-upload-form-100x100.jpg'
},
{
    url: 'http://tutorialzine.com/2013/04/services-chooser-backbone-js/',
    title: 'Your First Backbone.js App – Service Chooser',
    image: 'http://cdn.tutorialzine.com/wp-
content/uploads/2013/04/service_chooser_form-100x100.jpg'
}
];
```

```
}
```

```
/*-----
Simple reset
-----*/
```

```
*{
    margin:0;
    padding:0;
}
```

```
/*-----
General Styles
-----*/
```

```
body{
    font:15px/1.3 'Open Sans', sans-serif;
    color: #5e5b64;
    text-align:center;
}

a, a:visited {
    outline:none;
    color:#389dc1;
}

a:hover{
    text-decoration:none;
}

section, footer, header, aside, nav{
    display: block;
}

/*-----
    The search input
-----*/

.bar{
    background-color:#5c9bb7;

    background-image:-webkit-linear-gradient(top, #5c9bb7, #5392ad);
    background-image:-moz-linear-gradient(top, #5c9bb7, #5392ad);
    background-image:linear-gradient(top, #5c9bb7, #5392ad);

    box-shadow: 0 1px 1px #ccc;
    border-radius: 2px;
    width: 400px;
    padding: 14px;
    margin: 80px auto 20px;
    position:relative;
}
```

```
.bar input{
  background:#fff no-repeat 13px 13px;
  background-
image:url(data:image/png;base64,iVBORw0KGgoAAAANSUHEUgAAABAAAAAQCAyAAAAf8/
9hAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAyBpVFh0WE1MOmNv
bS5hZG9iZS54bXAAAAAADw/eHBhY2tldCBiZWdpbj0i77u/liBpZD0iVzVNME1wQ2VoaUh6c
mVTek5UY3prYzlkIj8+IDx4OnhtcG1ldGEgeG1sbnM6eD0iYWVrYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNzo
zMjowMCAglCAGlCAGl4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IiIe
G1sbnM6eG1wPSJodHRwOi8vb3MuYWRvYmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLnNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIHhtcDpDcmVhdG9yVGV9vbD0iQWRvY
mUgUGhvdG9zaG9wIENtNSBxZW5kb3dzLiB4bXBNTTpJbnN0YXV5ZjZUePSJ4bXAuaWlkOkU5N
EY0RTlFMtA4NzExRTM5RTEzQkFBQzMyRjkyQzVBliB4bXBNTTpEb2N1bWVudEIEPSJ4bXAuZG
lkOkU5NEY0RTlGMtA4NzExRTM5RTEzQkFBQzMyRjkyQzVBliB4bXBNTTpE1N0kRlcmI2ZWRGcm
9tIHNOUmVmOmIuc3RhbmNISUQ9InhtcC5paWQ6RTk0RjRFOUMxMDg3MTFFMzIFMTNcQU
FDMzJGOTJDNUeIiHN0UmVmOmRvY3VtZW50SUQ9InhtcC5kaWQ6RTk0RjRFOUQxMDg3MT
FFMzIFMTNcQUFDMzJGOTJDNUeIlz4gPC9yZGY6RGVzY3JpcHRpb24+IDwvcmlRmOIJERj4gPC
94OnhtcG1ldGE+IDw/eHBhY2tldCBiZmQ9InliPz4DjA/RAAABK0IEQVR42pTSQUdEURjG8dOY0
TqmPkGmRcqYD9CmzZAWJRHVRIa0iFYtM6uofYaiEW2SRJtEi9YxIklp07ZkWsww0v/wnByve7v
m5ee8M+85zz1jbt9Os+WiGkYdYxjCOx5wgFeXUHmtBSzpcCGa+5BJTCjEP+OnKWAT8xqe4ArP
GEEVC1hHEbs2oBwdXkM7mj/JLZrad437sCGHOfUtcziutuYu2v8XUFF/4f6vMK/YgAH1HxkBYV
60AR31gxkBYd6xAeF3VzMCwvzOBpypX8V4yuFRzX2d2gD/I5yjH4fYQEnzkj4fae5rJulF2sMXVr
AsaTWttRFu4Osb+1jEDT71/ZveyhouTch2fINQL9hKefKjuYFfuznXWzXMTabyrvfyIV3M4vhXgA
EAUMs7K0J9UJAAAAAASUVORK5CYII=);

  border: none;
  width: 100%;
  line-height: 19px;
  padding: 11px 0;

  border-radius: 2px;
  box-shadow: 0 2px 8px #c4c4c4 inset;
  text-align: left;
  font-size: 14px;
  font-family: inherit;
  color: #738289;
  font-weight: bold;
  outline: none;
```

```
        text-indent: 40px;
    }

    ul{
        list-style: none;
        width: 428px;
        margin: 0 auto;
        text-align: left;
    }

    ul li{
        border-bottom: 1px solid #ddd;
        padding: 10px;
        overflow: hidden;
    }

    ul li img{
        width:60px;
        height:60px;
        float:left;
        border:none;
    }

    ul li p{
        margin-left: 75px;
        font-weight: bold;
        padding-top: 12px;
        color:#6e7a7f;
    }
```

Navigation Menu

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8"/>
    <title>Learn AngularJS - Navigation Menu</title>

    <link href="http://fonts.googleapis.com/css?family=Open+Sans:400,700"
rel="stylesheet" />

    <!-- The main CSS file -->
    <link href="style.css" rel="stylesheet" />

    <!--[if lt IE 9]>
      <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
  </head>

  <!-- The ng-app directive tells angular that the code below should be evaluated -->

  <body ng-app>

    <!-- The navigation menu will get the value of the "active" variable as a class.
    The $event.preventDefault() stops the page from jumping when a link
is clicked. -->

    <nav class="{{active}}" ng-click="$event.preventDefault()">

      <!-- When a link in the menu is clicked, we set the active variable -->

      <a href="#" class="home" ng-click="active='home'">Home</a>
      <a href="#" class="projects" ng-click="active='projects'">Projects</a>
      <a href="#" class="services" ng-click="active='services'">Services</a>
      <a href="#" class="contact" ng-click="active='contact'">Contact</a>
    </nav>

    <!-- ng-show will show an element if the value in the quotes is truthful,
```


while ng-hide does the opposite. Because the active variable is not set

initially, this will cause the first paragraph to be visible. -->

```
<p ng-hide="active">Please click a menu item</p>
<p ng-show="active">You chose <b>{{active}}</b></p>

<!-- Include AngularJS from Google's CDN -->
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
</body>
</html>
```

```
/*-----  
    Simple reset  
-----*/
```

```
*{  
    margin:0;  
    padding:0;  
}
```

```
/*-----  
    General Styles  
-----*/
```

```
body{  
    font:15px/1.3 'Open Sans', sans-serif;  
    color: #5e5b64;  
    text-align:center;  
}
```

```
a, a:visited {  
    outline:none;  
    color:#389dc1;  
}
```

```
a:hover{  
    text-decoration:none;  
}
```

```
section, footer, header, aside, nav{  
    display: block;  
}
```

```
/*-----  
    The menu  
-----*/
```

```
nav{
    display:inline-block;
    margin:60px auto 45px;
    background-color:#5597b4;
    box-shadow:0 1px 1px #ccc;
    border-radius:2px;
}

nav a{
    display:inline-block;
    padding: 18px 30px;
    color:#fff !important;
    font-weight:bold;
    font-size:16px;
    text-decoration:none !important;
    line-height:1;
    text-transform: uppercase;
    background-color:transparent;

    -webkit-transition:background-color 0.25s;
    -moz-transition:background-color 0.25s;
    transition:background-color 0.25s;
}

nav a:first-child{
    border-radius:2px 0 0 2px;
}

nav a:last-child{
    border-radius:0 2px 2px 0;
}

nav.home .home,
nav.projects .projects,
nav.services .services,
nav.contact .contact{
    background-color:#e35885;
}

p{
    font-size:22px;
```

```
    font-weight:bold;
    color:#7d9098;
}

p b{
    color:#ffffff;
    display:inline-block;
    padding:5px 10px;
    background-color:#c4d7e0;
    border-radius:2px;
    text-transform:uppercase;
    font-size:18px;
}
```

Order Form

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8"/>
    <title>Learn AngularJS - Order Form</title>

    <link
href="http://fonts.googleapis.com/css?family=Cookie|Open+Sans:400,700" rel="stylesheet"
/>

    <!-- The main CSS file -->
    <link href="style.css" rel="stylesheet" />

    <!--[if lt IE 9]>
      <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
  </head>

  <!-- Declare a new AngularJS app and associate the controller -->
  <body ng-app ng-controller="OrderFormController">

    <form>
```

```
<h1>Services</h1>

<ul>
  <!-- Loop through the services array, assign a click handler, and
set or
      remove the "active" css class if needed -->
  <li ng-repeat="service in services" ng-
click="toggleActive(service)" ng-class="{active:service.active}">
    <!-- Notice the use of the currency filter, it will format
the price -->
    {{service.name}} <span>{{service.price |
currency}}</span>
  </li>
</ul>

<div class="total">
  <!-- Calculate the total price of all chosen services. Format it as
currency. -->
  Total: <span>{{total() | currency}}</span>
</div>

</form>

<!-- Include AngularJS from Google's CDN -->
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
<script src="script.js"></script>
</body>
</html>
```

```
function OrderFormController($scope){
```

```
  // Define the model properties. The view will loop
  // through the services array and generate a li
  // element for every one of its items.
```

```
$scope.services = [
  {
    name: 'Web Development',
    price: 300,
    active:true
  },{
    name: 'Design',
    price: 400,
    active:false
  },{
    name: 'Integration',
    price: 250,
    active:false
  },{
    name: 'Training',
    price: 220,
    active:false
  }
];

$scope.toggleActive = function(s){
  s.active = !s.active;
};

// Helper method for calculating the total price

$scope.total = function(){
  var total = 0;

  // Use the angular forEach helper method to
  // loop through the services array:

  angular.forEach($scope.services, function(s){
    if (s.active){
      total+= s.price;
    }
  });

  return total;
};
```

```
}
```

```
/*-----  
    Simple reset  
-----*/
```

```
*{  
    margin:0;  
    padding:0;  
}
```

```
/*-----  
    General Styles  
-----*/
```

```
body{  
    font:15px/1.3 'Open Sans', sans-serif;  
    color: #5e5b64;  
    text-align:center;  
}
```

```
a, a:visited {  
    outline:none;  
    color:#389dc1;  
}
```

```
a:hover{  
    text-decoration:none;  
}
```

```
section, footer, header, aside, nav{  
    display: block;  
}
```

```
/*-----  
    The order form
```

-----*/

```
form{
  background-color: #61a1bc;
  border-radius: 2px;
  box-shadow: 0 1px 1px #ccc;
  width: 400px;
  padding: 35px 60px;
  margin: 80px auto;
}

form h1{
  color:#fff;
  font-size:64px;
  font-family:'Cookie', cursive;
  font-weight: normal;
  line-height:1;
  text-shadow:0 3px 0 rgba(0,0,0,0.1);
}

form ul{
  list-style:none;
  color:#fff;
  font-size:20px;
  font-weight:bold;
  text-align: left;
  margin:20px 0 15px;
}

form ul li{
  padding:20px 30px;
  background-color:#e35885;
  margin-bottom:8px;
  box-shadow:0 1px 1px rgba(0,0,0,0.1);
  cursor:pointer;
}

form ul li span{
  float:right;
}
```



```
form ul li.active{
    background-color:#8ec16d;
}
```

```
div.total{
    border-top:1px solid rgba(255,255,255,0.5);
    padding:15px 30px;
    font-size:20px;
    font-weight:bold;
    text-align: left;
    color:#fff;
}
```

```
div.total span{
    float:right;
}
```

Switchable - Grid

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8"/>
        <title>Learn AngularJS - Switchable Grid</title>

        <link
href="http://fonts.googleapis.com/css?family=Cookie|Open+Sans:400,700" rel="stylesheet"
/>

        <!-- The main CSS file -->
        <link href="style.css" rel="stylesheet" />

        <!--[if lt IE 9]>
            <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
        <![endif]-->
```

```
</head>

<body ng-app="switchableGrid" ng-controller="SwitchableGridController">

  <div class="bar">

    <!-- These two buttons switch the layout variable,
           which causes the correct UL to be shown. -->

    <a href="#" class="list-icon" ng-class="{active: layout == 'list'}" ng-
click="layout = 'list'"></a>
    <a href="#" class="grid-icon" ng-class="{active: layout == 'grid'}" ng-
click="layout = 'grid'"></a>
  </div>

  <!-- We have two layouts. We choose which one to show depending on the
"layout" binding -->

  <ul ng-show="layout == 'grid'" class="grid">
    <!-- A view with big photos and no text -->
    <li ng-repeat="p in pics">
      <a href="{{p.link}}" target="_blank"></a>
    </li>
  </ul>

  <ul ng-show="layout == 'list'" class="list">
    <!-- A compact view smaller photos and titles -->
    <li ng-repeat="p in pics">
      <a href="{{p.link}}" target="_blank"></a>
      <p>{{p.caption.text}}</p>
    </li>
  </ul>

  <!-- Include AngularJS from Google's CDN and the resource module -->
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular-
resource.min.js"></script>
  <script src="script.js"></script>
```

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

```
// Define a new module. This time we declare a dependency on
// the ngResource module, so we can work with the Instagram API
```

```
var app = angular.module("switchableGrid", ['ngResource']);
```

```
// Create and register the new "instagram" service
app.factory('instagram', function($resource){
```

```
    return {
        fetchPopular: function(callback){
```

```
            // The ngResource module gives us the $resource service. It makes
            working with
```

```
            // AJAX easy. Here I am using a client_id of a test app. Replace it with
            yours.
```

```
            var api =
            $resource('https://api.instagram.com/v1/media/popular?client_id=:client_id&callback=JSON_CALLBACK',{
```

```
                client_id: '642176ece1e7445e99244cec26f4de1f'
            },{
```

```
                // This creates an action which we've chosen to name "fetch".
            It issues
```

```
                // an JSONP request to the URL of the resource. JSONP
            requires that the
```

```
                // callback=JSON_CALLBACK part is added to the URL.
```

```
                fetch:{method:'JSONP'}
            });
```

```
            api.fetch(function(response){
```

```
                // Call the supplied callback function
                callback(response.data);
```

```
        });
    }
}

});

// The controller. Notice that I've included our instagram service which we
// defined below. It will be available inside the function automatically.

function SwitchableGridController($scope, instagram){

    // Default layout of the app. Clicking the buttons in the toolbar
    // changes this value.

    $scope.layout = 'grid';

    $scope.pics = [];

    // Use the instagram service and fetch a list of the popular pics
    instagram.fetchPopular(function(data){

        // Assigning the pics array will cause the view
        // to be automatically redrawn by Angular.
        $scope.pics = data;
    });
}

/*-----
   Simple reset
-----*/

*{
    margin:0;
    padding:0;
}
```

```
/*-----  
    General Styles  
-----*/
```

```
body{  
    font:15px/1.3 'Open Sans', sans-serif;  
    color: #5e5b64;  
    text-align:center;  
}
```

```
a, a:visited {  
    outline:none;  
    color:#389dc1;  
}
```

```
a:hover{  
    text-decoration:none;  
}
```

```
section, footer, header, aside, nav{  
    display: block;  
}
```

```
/*-----  
    The search input  
-----*/
```

```
.bar{  
    background-color:#5c9bb7;  
  
    background-image:-webkit-linear-gradient(top, #5c9bb7, #5392ad);  
    background-image:-moz-linear-gradient(top, #5c9bb7, #5392ad);  
    background-image:linear-gradient(top, #5c9bb7, #5392ad);  
  
    box-shadow: 0 1px 1px #ccc;  
    border-radius: 2px;  
    width: 580px;  
    padding: 10px;
```

```
margin: 80px auto 25px;
position: relative;
text-align: right;
line-height: 1;
}

.bar a {
background: #4987a1 center center no-repeat;
width: 32px;
height: 32px;
display: inline-block;
text-decoration: none !important;
margin-right: 5px;
border-radius: 2px;
}

.bar a.active {
background-color: #c14694;
}

.bar a.list-icon {
background-
image: url(data:image/png;base64,iVBORw0KGgoAAAANSUHEUgAAABAAAAAQCAAAAAf8/
9hAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAyBpVFh0WE1MOmNv
bS5hZG9iZS54bXAAAAAADw/eHBhY2tldCBiZWdpbj0i77u/liBpZD0iVzVNME1wQ2VoaUh6c
mVTek5UY3prYzlkIj8+IDx4OnhtcG1ldGEgeG1sbnM6eD0iYWVrYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNzo
zMjowMCAgICAgICAgIj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3NpYmU6YWRvYmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLnNvbS94YXAvMS4wL211LjIgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJjZVJlZiMiIlHhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSB3aW5kb3dzLiB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOkYzN
kFCQ0ZBMTBCRTErRTM5NDk4RDFEM0E5RkQ1NEZCIB4bXBNTTpEb2N1bWVudEIEPSj4bXAu
ZGllOkYzNkFCQ0ZCMTBCRTErRTM5NDk4RDFEM0E5RkQ1NEZCIB4gPHhtcE1NOkRlcml2ZWR
Gcm9tIHNOUmVmOmRm3RhbmNlSUQ9InhtcC5paWQ6RjM2QUJDRjgxMEJFMtFFMzk00ThE
MUQzQTIGRDU0RklilHN0UmVmOmRvY3VtZW50SUQ9InhtcC5kaWQ6RjM2QUJDRjgxMEJFM
TFFMzk00ThEMUQzQTIGRDU0RklilLz4gPC9yZGY6RGVzY3JpcHRpb24+IDwvcmlRmOIJERj4gPC
94OnhtcG1ldGE+IDw/eHBhY2tldCBibmQ9InliPz7h1bLqAAAAWUUEQVR42mL8/////BwYGBn4G
CACxBRlIAxAA/4jaXoPEkMyjJ+A/g9MDJQBRhYg8RFqMwg8RJIUINYLFDMBUi+ADQAF1n8ofk
9yIAy6WPg4GgtDMRYAAgWAdLYwLAolwPgAAAAASUVORK5CYII=);
```

}

```
.bar a.grid-icon{
    background-
image:url(data:image/png;base64,iVBORw0KGgoAAAANSUUhEUgAAABAAAAQCAAAAAf8/
9hAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAyBpVFh0WE1MOmNv
bS5hZG9iZS54bXAAAAAADw/eHBhY2tldCBiZWdpbj0i77u/liBpZD0iVzVNME1wQ2VoaUh6c
mVTek5UY3prYzlkIj8+IDx4OnhtcG1ldGEgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWxuczp4bXBNTT0iaHR0c
DovL25zLmFkb2JlLmNvbS94YXAvMS4wL21tLylgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG
9iZS5jb20veGFwLzEuMC9zVHlwZS9ScXNvdXJzVjVJZiMiIlhtcDpDcmVhdG9yVG9vbD0iQWRvY
mUgUGhvdG9zaG9wIENTNSBXaW5kb3dzliB4bXBNTTpJbnN0YXV5ZGUlEPSj4bXAuaWlkOjBEQ
kMyQzE0MTBCRjExRTNBMDIj4gPHJkZjpSREYgeG1sbnM6eD0iYWVhYmU6bnM6bWV0YS8iIHg6eG
1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLjEzNDc3NywgMjAxMC8wMi8xMi0xNz0
zMjowMCAglCAGlCAGlj4gPHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xO
Tk5LzAyLzlyLXJkZi1zeW50YXgtbnMjIj4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IilgeG1sb
nM6eG1wPSJodHRwOi8vb3N0LmUuY29tL3h5cC8xLjAvliB4bWx
```

```
.bar input{
    background:#fff no-repeat 13px 13px;

    border: none;
    width: 100%;
    line-height: 19px;
    padding: 11px 0;

    border-radius: 2px;
    box-shadow: 0 2px 8px #c4c4c4 inset;
    text-align: left;
    font-size: 14px;
    font-family: inherit;
    color: #738289;
    font-weight: bold;
    outline: none;
```

```
        text-indent: 40px;
    }
```

```
/*-----
    List layout
-----*/
```

```
ul.list{
    list-style: none;
    width: 500px;
    margin: 0 auto;
    text-align: left;
}
```

```
ul.list li{
    border-bottom: 1px solid #ddd;
    padding: 10px;
    overflow: hidden;
}
```

```
ul.list li img{
    width:120px;
    height:120px;
    float:left;
    border:none;
}
```

```
ul.list li p{
    margin-left: 135px;
    font-weight: bold;
    color:#6e7a7f;
}
```

```
/*-----
    Grid layout
-----*/
```



```
ul.grid{  
    list-style: none;  
    width: 570px;  
    margin: 0 auto;  
    text-align: left;  
}
```

```
ul.grid li{  
    padding: 2px;  
    float:left;  
}
```

```
ul.grid li img{  
    width:280px;  
    height:280px;  
    display:block;  
    border:none;  
}
```