Introduction to Angular.js

Angular

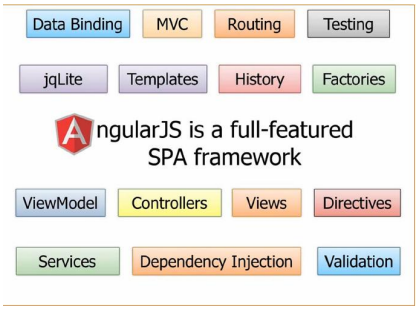
- Application platform by itself.
- Now, API centric appl.
  - Server load is coming down.
  - HAT stack: html, Angular, Thin server.
- New gen servers:
  - service oriented
  - API centric

Angular

- MVC based application:
  - Extendable
  - Maintainable
  - Testable
  - Standardized

Client-Side Processing

- Earlier, wrt multi-page / SPA web applications:
  - Server created the HTML by assembling and merging it with data.
- But, in Angular:
  - The template and data get shipped to the browser to be assembled there.



HTML File

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Angular JS</title>
    <script src="../libs/angular.min_v1.2.4.js"></script>
    <script src="./controller.js"></script>

</head>

<body>
    <div ng-controller='HelloController'>
            <p>{{greeting.text}}, World</p>
    </div>

</body>
</html>
```


Controller

```
function  HelloController($scope) {
    $scope.greeting = { text: 'Hello' };
}
```

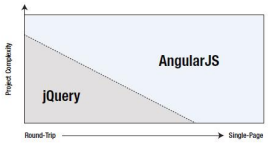

Note …

- There are no classes or IDs in the HTML to identify where to attach event listeners.
- HelloController set the greeting.text to “Hello”.
  - But no need to register any event listeners or callbacks.
- HelloController is a plain JavaScript class.
  - Doesn’t inherit from anything that Angular provides.
- HelloController got the $scope object that it needed without having to create it.
  - No need to call the HelloController’s constructor ourselves, or figure out when to call it.

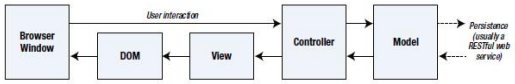
Model View Controller (MVC)

- MVC application structure was introduced in the 1970s,
- Clear separation in the code between:
  - managing its data (model),
  - the application logic (controller).
  - presenting the data to the user (view).

Angular



The MVC pattern @ Angular



Controller

- A controller built using the MVC *should*
  - Contain the logic required to initialize the scope
  - Contain the logic/behaviours required by the view to present data from the scope
  - Contain the logic/behaviours required to update the scope based on user interaction
- The controller *should not*
  - Contain logic that manipulates the DOM (that is the job of the view)
  - Contain logic that manages the persistence of data (that is the job of the model)
  - Manipulate data outside of the scope

Model

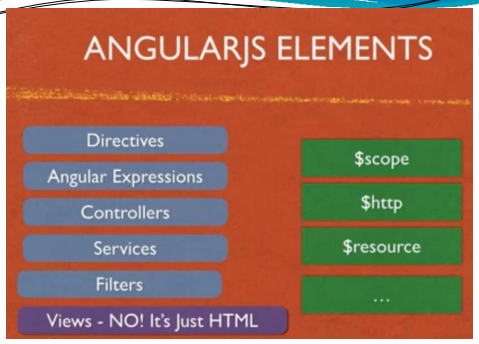
- The model in an application built using the MVC pattern *should*
  - Contain the domain data
  - Contain the logic for creating, managing, and modifying the domain data (& using web services)
  - Provide a clean API that exposes the model data and operations on it.
- The model *should not*
  - Expose details of how the model data is obtained or managed
    - Details of the data storage mechanism or the remote web service should not be exposed to controllers and views.
  - Contain logic that transforms the model based on user interaction (its controller's job)
  - Contain logic for displaying data to the user (the view's job)

View

- Views *should*
  - Contain the logic and markup required to present data to the user
- Views *should not*
  - Contain complex logic (better placed in a controller)
  - Contain logic that creates, stores, or manipulates the domain model.

Summary

- View logic should prepare data only for display and never modify the model.
- Controller logic should never directly create, update, or delete data from the model.
- The client should never directly access the data store.



Data Binding

- Earlier approach:
  - Merge template HTML strings with data.
  - Update DOM by setting innerHtml on a placeholder element.
- Now:
  - Declare which parts of the UI map to which JavaScript properties.
  - They sync automatically.

Using Directives and Data Binding Syntax

```
<!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>
```

Directive

Directive

Data Binding Expression

Common Directives

- Just extend HTML's syntax.
  - Several new attributes in the templates that aren't part of the HTML.
- <html ng-app>
- <body ng-controller='MyController'>
-
- <input ng-model='item.quantity'>
- Filter: transforms the text.
- <button ng-click='remove($index)'>Remove</button>

Anatomy of an AngularJS Application

Invoking Angular

- Any application must do two things to start Angular:
  - Load the angular.js library
  - Tell Angular which part of the DOM it should manage with the ng-app directive.
- Loading:
  - Download & host locally.
  - CDN

Declaring Angular's Boundaries

- When building an all-Angular application:

```
<html ng-app>
...
</html>
```

- To manage only a **part** of the page:

```
<html>
...
  <div ng-app>
  ...
  </div>
...
</html>
```


Model, View ←→Controller

- In the app:
  - A model containing data that represents the current state of the application.
  - Views that display this data.
  - Controllers that manage the relationship between the model and the views.
- Template:

```
<p>{{someText}}</p>
```


View, Controller, Scope

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

Templates and Data Binding

```
<div   ng-repeat="item in items">
    <span>{{item.title}}</span>
    ...
</div>
```


Displaying Text

Two equivalent ways:

- `<p>{{greeting}}</p>`
- `<p   ng-bind="greeting"></p>`

Form Inputs

- The *ng-model* attribute binds elements to the model properties.
  - This works with all the standard form elements like text inputs, radio buttons, checkboxes, etc.
- Use *ng-submit* directive on the form itself, to specify a function to call when the form submits (i.e. user activated).

Other Events

- ng-click
- ng-dblclick

```
<div class="navbar" ng-controller="NavController">
…
  <li class="menu-item" ng-click="doSomething()">Something</li>
…
</div>
<div class="contentArea" ng-controller="ContentAreaController">
…
  <div ng-click="doSomething()">...</div>
…
</div>


 function NavController($scope) {
   $scope.doSomething = doA;
 }

 function ContentAreaController($scope) {
   $scope.doSomething = doB;
 }
```


Lists, Tables, and Other Repeated Elements

- The *ng-repeat* directive gives a reference to the index of the current element via *$index*.
- $first, $middle, and $last

Formatting Data with Filters

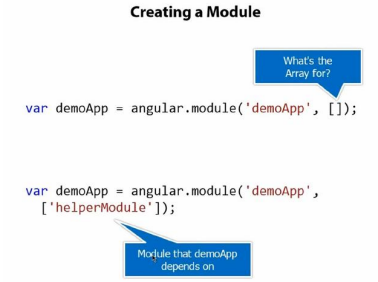
- Helps transform data for display to the user within an interpolation in the template.

  {{ expression | filterName : parameter1 : …parameterN }}
- Common filters:
  - currency
  - date
  - number
  - uppercase
  - lowercase

Dependency Injection

- The $scope object that does our data binding is passed on automatically.
- No need to create it by calling any function.
  - Just asked for it by putting it in HelloController's constructor.

Creating a Module



Use of Modules

- Keeps the global namespace clean.
- Makes tests easier to write & execute - target isolated functionality.
- Easier to share code between applications.
- Allows the app to load different parts of the code in any order.

Modules

- When declaring a module, two parameters need to be passed:

  angular.module("myApp", []) ;

- To get a reference to this module (getter method):

  angular.module("myApp") ;

Organizing Dependencies with Modules

- Controller: has code that exposes the right data and functions to the view template.
  - Dumping ground for anything we need to do.
  - Becomes hard to understand and change.
- Modules: provide a way to group dependencies for a functional area within the application.
  - Uses a mechanism to automatically resolve dependencies (dependency injection).

```
var appMod = angular.module('app', ['MyHelperWidgets',
                                    MyDatSync'];
```

Separating UI Responsibilities with Controllers

Controllers have three responsibilities:

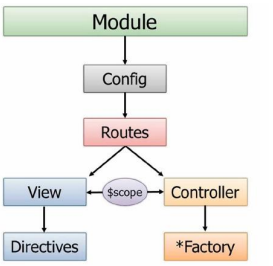
- Set up the initial state in the application's model.
- Expose model and functions to the view (UI template) through $scope.
- Watch other parts of the model for changes and take action.

$watch

- Observing Model Changes
  - Notifies when parts of the model change.
  - Watch individual object properties and computed results (functions).

$watch(watchFn, watchAction, deepWatch)

The Framework



Benefits

- **AngularJS provides a robust "SPA" framework for building robust client-centric applications**
- **Key features:**
  - Directives and filters
  - Two-way data binding
  - Views, Controllers, Scope
  - Modules and Routes

The 5 D's

a) Declarative - extended HTML vocabulary.
b) Dataflow / data-binding.
c) DRY
d) Dependency Injection: decoupled components / modules.
e)Designer friendly: HTML based.

5 Commandments of Angular

a) HTML is the view.
b) REST APIs should provide PERFECT JSON.
c) Communication is one-way: Directives -> HTML -> Controller -> Services.
d) The controller does not manipulate the DOM (use directives).
e) Single responsibility principle for controllers, services & directives.

Communicating with Servers

$http

Traditional Ajax (XHR):

- Get a handle on the XMLHttpRequest object.
- Make the request
- Read the response
- Check the error codes
- Process the server response.

Ajax API

```
var  xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function() {
      //response is ready
    if (xmlhttp.readystate == 4 && xmlhttp.status == 200) {
        var response = xmlhttp.responseText;
    } else if (xmlhttp.status == 400) {  // 4xx  series
       // Handle error gracefully
    }
};
xmlhttp.open("GET", "http://myserver/api", true);
xmlhttp.send();
```


AngularJS XHR API: GET

```
$http.get('api/user', {params: {id: '5'}
}).success(function(data, status, headers, config) {
    // Do something successful.
}).error(function(data, status, headers, config) {
   // Handle the error
});
```


POST

```
var postData = {text: 'long blob of text'};
// The next line gets appended to the URL as params
// so it would become a post request to /api/user?id=5
var config = {params: {id: '5'}};

$http.post('api/user', postData, config
    ).success(function(data, status, headers, config) {
      // Do something successful
    }).error(function(data, status, headers, config) {
      // Handle the error
});
```