Volume 5, No. 12, December 2014
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICLE
Available Online at www.jgrcs.info

Nilesh Jain[1], Priyanka Mangal[2] and Deepak Mehta[3]

MCA, Lecturer[1, 3]

Mandsaur Institue of Technology[1,2, 3]

CSE,Lecturer[2]

Email: nilesh.jain@mitmandsaur.info[1]

AngularJS: A Modern MVC Framework in JavaScript

Abstract: AngularJS is a JavaScript MVC Framework created by Google to build properly architecture and maintainable web application. AngularJS is built around the philosophy that declarative code is better than imperative code while building UIs and wiring different components of web applications together. In this article we have shown the features of AngularJS.

INTRODUCTION

AngularJS is not a library rather AngularJS is a JavaScript framework that embraces extending HTML into a more expressive and readable format. It allows you to decorate your HTML with special markup that synchronizes with your JavaScript leaving you to write your application logic instead of manually updating views. Whether you're looking to augment existing JavaScript applications or harness the full power of the framework to create rich and interactive SPA's, Angular can help you write cleaner and more efficient code.

This one may seem obvious, but it's important to remember that many (not all) frameworks are made by hobbyists in the open source community. While passion and drive have forged frameworks, like **Cappucino** and **Knockout**, Angular is built and maintained by dedicated (and highly talented) Google engineers. This means you not only have a large open community to learn from, but you also have skilled, highly-available engineers tasked to help you get your Angular questions answered.

This isn't Google's first attempt at a JavaScript framework; they first developed their comprehensive Web Toolkit, which compiles Java down to JavaScript, and was used by the Google Wave team extensively. With the rise of HTML5, CSS3, and JavaScript, as both a front-end and back-end language, Google realized that the web was not meant to be written purely in Java.

Why Choose AngularJS?

1. DOM has markup [**Angular markup lives in the DOM]**
2. Data is POJO [**Angular uses plain old JavaScript objects**]
3. DI for modules [**Angular heavily leverages Dependency Injection**]

1. **DOM has markup:**

```
<!DOCTYPE html>
<html>
<head>
    <script src="/js/angular.js"></script>
    <script>
        var app = angular.module('MyApp', []);
    </script>
</head>
<body ng-app="MyApp">

    <input type="text" ng-model="someval" />

    <my-custom-tag>{{ someval }}</my-custom-tag>

</body>
</html>
```

Templates in most client-side JavaScript frameworks work like something like this:

- template with markup -> framework template engine -> HTML -> DOM

Angular, on the other hand puts markup directly into the HTML document and the flow looks like this:

- HTML with Angular markup -> DOM -> Angular template engine

Angular evaluates the markup only after HTML has been loaded into the DOM.

This approach has three major benefits.

1. **Integration with Existing Apps** - Since Angular only starts evaluating the page at the end of the loading process (i.e. once HTML is in the DOM), it is very easy to sprinkle small bits of Angular "magic" on top of existing applications.
2. **Simplicity** - You can work with Angular in a basic HTML document from you local file system. Just open the HTML document in your browser. No need for any web server or template build process. I have found this very useful for creating quick mockups of a new website or piece of functionality.
3. **Extensibility** - Using Directives, Angular allows you to create custom elements and attributes that extend the standard HTML vocabulary. For example, in this slide there is a my-custom-tag element. Using Angular you can define how that element is rendered and assign behaviors to it. This allows you to create a library of your own reusable components.

Data is POJO:

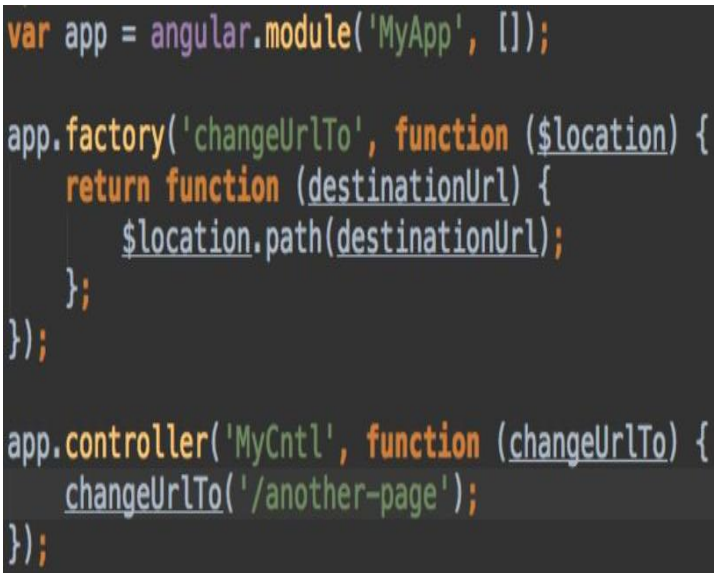
```
    <script>
        var app = angular.module('MyApp', []);
        app.controller('MyCntl', function ($scope) {
            $scope.guys = ['jeff', 'joe', 'bob'];
        })
    </script>
</head>
<body ng-app="MyApp">
    <ul ng-controller="MyCntl">
        <li ng-repeat="guy in guys">{{ guy }}</li>
    </ul>
</body>
```

Angular is one of the only major front end frameworks that utilize plain old Javascript objects (POJOs) for the model layer. This makes it extremely easy to integrate with existing data sources and play with basic data.

Let's say you make an AJAX call to get some data from an API. Before you can bind that data to the DOM, most frameworks require you to wrap the data in Model objects that have getters and setters. The getters/setters are how non-Angular frameworks propagate data change events.

Angular gets around this by using a process called dirty checking where snapshots of data over time are compared to see if anything has changed. While there are certainly some downsides to this approach (ex. $scope.$apply, data binding limits, etc.)

3. DI for modules:



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

app.factory('changeUrlTo', function ($location) {
    return function (destinationUrl) {
        $location.path(destinationUrl);
    };
});

app.controller('MyCntl', function (changeUrlTo) {
    changeUrlTo('/another-page');
});
```

There are some people that love dependency injection and there are some people that hate it. If you are going to work with Angular, you sort of need to be in the former camp. I personally love it because it promotes better modularization of code and enables strong unit testing.

Unit testing front end code is usually hard because there are so many sticky dependencies. Angular's DI allows you to mock out many of these dependencies and isolate individual components.

FEATURES OF 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 are 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">
      <img ng-src="{{image.thumbnail}}"
alt="{{image.description}}">
    </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.

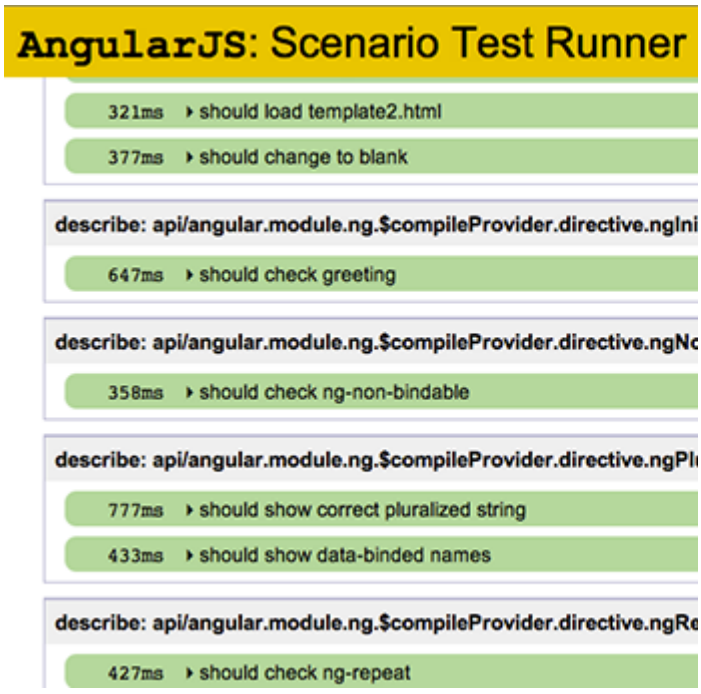
BONUS FEATURE: TESTING

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:



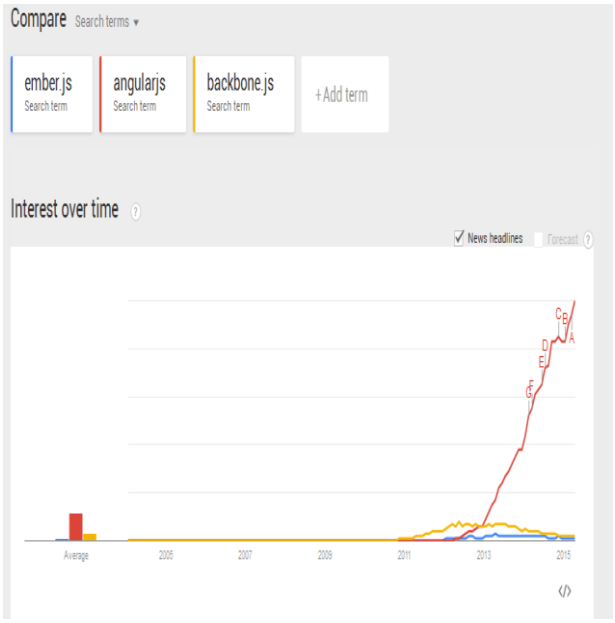
The API documentation is full of end-to-end tests that do an incredible job of illustrating how a certain part of the framework should work. After a while, I found myself going straight to the tests to see how something worked, and then maybe reading the rest of the documentation to figure something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

Comparisons in Community:

Community is one of the most important factors to consider when choosing a framework. A large community means more questions answered, more third-party modules, more YouTube tutorials…you get the point. I have put together a table with the numbers, as of August 16, 2014. Angular is definitely the winner here, being the 6th most-starred project on GitHub and having more questions on StackOverflow than Ember and Backbone combined, as you can see below:

Metric	AngularJS	Backbone.js	Ember.js
Stars on Github	27.2k	18.8k	11k
Third-Party Modules	800 ngmodules	236 backplugs	21 emberaddons
StackOverflow Questions	49.5k	15.9k	11.2k
YouTube Results	~75k	~16k	~6k
GitHub Contributors	928	230	393
Chrome Extension Users	150k	7k	38.3k

All those metrics, however, merely show the current state of each framework. It is also interesting to see which framework has a faster-growing popularity. Fortunately, using Google Trends (Till 18/4/2015) we can get an answer for that too:



ngularjs-for-your-next-project--net-28457

CONCLUSION:

Angular's innovative approach for extending HTML will make a lot of sense for people who are web developers in soul. With a large community and Google behind it, it is here to stay and grow, and it works well both for quick prototyping projects and large-scale production applications.

REFERENCES:

[1] AngularJS vs. Backbone.js vs. Ember.js (https://www.airpair.com/js/javascript-framework-comparison)

[2] https://www.airpair.com/angularjs/posts/jquery-angularjs-comparison-migration-walkthrough

[3] http://code.tutsplus.com/tutorials/3-reasons-to-choose-a