WHITE PAPER

Enriching UI with AngularJS

aspire SYSTEMS
attention. always.

CONTENTS

AngularJS architecture allows crafting sophisticated and scalable UI

Executive Summary

This paper discusses AngularJS for developing Interactive Web Application interacting with web services. Angular JS architecture allows crafting sophisticated and scalable UI with great ease with its structured framework design and vast collections of UI widgets.

Motivation to use JavaScript for Interactive Web Application

- Application demanding a decoupled view and data wherein the data and views can be independently changed without impacting each other.
- Need for supporting additional, removal and change of views using the same model by increasing the flexibility for custom UI creation.
- The presentation/view rendering logic has to be distributed to the less loaded end-userbrowsers so that server load can be dedicated toprocess data or computation intensive activities. Also, static contents like html, CSS, images and media files can be moved to CDN (Content Delivery Network) servers making reduced load on servers.
- All common user inputs validation like data type verification and data format verification can be independently handled by the JavaScript.

These characteristics of JavaScript makes AngularJSa commonly adopted framework by the community for creating Interactive Web Application with rich and flexible user interface independent of business services hosted from servers.

AngularJS Strengths

Modularity: AngularJS is a modularized framework. There is a clear separation of server side business logic, data model and UI presentation logic. This helps in faster development cycles and easier maintenance.

Easy Development Cycles: AngularJS comes as addendum to the HTML tag attributes. This ensures easily learning and faster development cycles.

Widgets and Event Handlers: AngularJS has got rich collection of widgets which can be used in building sophisticated UI with greater ease. This also has got a simpler event handling mechanism to track any browser based events or user interactions. For many, complex UI presentation logic like cumbersome charting logic or visualization algorithm realization, plenty of free or paid plugins are available.

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies. The Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.

There are only three ways a component (object or function) can get a hold of its dependencies:

- The component can create the dependency, typically using the new operator.
- The component can look up the dependency, by referring to a global variable.
- The component can have the dependency passed to it where it is needed.

Automated Test Frameworks Unit testing involves breaking down the logic of your application into small chunks or 'units', and verifying that each unit works as desired. The core units which make up features should be verified with accompanying unit tests. In JavaScript apps, the smallest units of code you can test are usually individual functions. We use such AngularJS function written in JavaScript using wide range of unit testing frameworks like Jasmine, Mocha, QUnit. With these frameworks we can write unit test scripts to ensure functional completeness with every releases. There are many environments like Karma or CasperJS which helps to run these test scripts to decide on the behaviour.

The core units which make up features should be verified with accompanying unit tests.

Single Page Application (SPA): AngularJS is a strong SPA framework supporting features like routing, back button support, view composition/dynamically loading of partial views/templates, client side validation, etc... for anything larger.

Cross Browser Support: AngularJS uses bootstrap framework for UI presentation. This is referred as ui.bootstrap module. This is a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. This bootstrap will be compatible with all major browser's latest version. So AngularJS scripts is highly supported across all browsers' latest versions.

AngularJS is very much powerful in building Single Page Application (SPA)

AngularJS for SPA

AngularJS is very much powerful in building **Single Page Application** (SPA). Its routing and template supports helps to develop SPA easily and also their maintenance is simpler.

The key design steps in creating SPA with Angular JS:

- Identify single page design elements including different views
- Identify one or more controllers that would be used with these views
- Glue views and controllers together with Angular routing

Routing functionality provided by Angular helps you to create a single page app where multiple views could be loaded in the single page based on the URL parameter.

The routing functionality is provided by Angular **ngRoute**module. Application routes in Angular are declared via the **$routeProvider**, which is the provider of the $route service. This service makes it easy to wire together controllers, view templates, and the current URL location in the browser.

Following are simple samples for SPA. This has got a main page as landing page and two pages that can be navigated from the landing page.

Sample main page

```
<html ng-app="helloApp">
<head>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.18/angular.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.18/angular-
route.js"></script>
<!--script src="myangular.js"></script-->
<script>
varhelloApp = angular.module("helloApp", ['ngRoute']);

helloApp.config(function($routeProvider){
$routeProvider
.when('/home', { controller: 'CompanyCtrl', templateUrl: 'home.html' } )
.when('/about', { controller: 'CompanyCtrl', templateUrl: 'about.html' })
.otherwise( { redirectTo: '/home' } );
});
</script>
</head>
<body>
This is a simple SPA where you can traverse between two pages <ul><li><a
href="#/home">Home </a>and </li>
<li><a href="#/about">About </a></li></ul>
<P>
<div ng-view=""></div>
</body>
</html>
```

Two pages home.html and about.html that can be reached from main landing page

Home.html

```
<html>
<body>
<H1> this is home page </H1>
</body>
</html>
```

about.html

```
<html>
<body>
<H1> this is about page </H1>
</body>
</html>
```

In the given example

- Directive ng-app="helloApp": Defines the HTML page as angular module namely, helloApp
- Directive ng-view which is used to load different views based on routing configuration
- In the script we have mentioned how routing has to be carried out

Templates 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 data-bindings are DOM transformations, not string concatenations or inner HTML 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 for using the ng-repeat directive to loop over the images array and populate data.

```
<!DOCTYPE html>
<html>
<body>
<div ng-app="" ng-controller="monthloop">
<p>Looping with objects:</p>
<ul>
<li ng-repeat="x in months">  {{ x }}  </li>
</ul>
</div>
<script>
functionmonthloop($scope) {
   $scope.months= [      'January','February','March','April'   ];
}
</script>
<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"></script>
</body>
</html>
```

AgnularJS for HTML5

AngularJS adheres highly to HTML5 standards with MVVM Architecture.

This supports WebSockets, HTML5 based widgets, local storage etc.

Explicit Declaration In AngularJS, we can explicitly mention a web application as HTML5 application. An Example code fragment is given below

Code fragment that accesses the local storage to save intermediate results

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

app.config(function ($routeProvider, $locationProvider) {
  $locationProvider.html5Mode(true); // enable HTML 5 pushState
  $routeProvider.when('/', {
templateUrl: '/main.html',
controller: 'AppCtrl'
  });
  $routeProvider.when('/another', {
templateUrl: '/another.html',
controller: 'AppCtrl'
  });
});

app.controller('AppCtrl', function ($scope) {
    $scope.model = {
message: 'This is HTML 5 application!!!'
    };
});
```

HTML5 Widgets AngularJS uses HTML5 components and provides an extra thin layer of customization to provide two-way data binding. A simple example is given below which uses HTML5 based email component.

```
<!DOCTYPE html>
<html>
<body>
<div ng-app="">
<p>Input something in the input box:</p>
<p>Name: <input type="email" ng-model="name" value="John"></p>
<p ng-bind="name"></p>
</div>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>
</body>
</html>
```

Usage of Local Storage Following is a code fragment that accesses the local storage to save intermediate results

```
varlocalstorageApp = angular.module('localstorageApp', ['LocalStorageModule']);

localstorageApp.factory('stockInfo', ['$http', '$rootScope', function($http, $rootScope) {
varmystock = [];
return {
getStock: function() {
return $http.get(base_url + 'ajax/getStock').then(function(response) {
mystock = response.data;
      $rootScope.$broadcast('stockInfo',mystock);
returnmystock;
    })
  }
 };
}]);
```

```
localstorageApp.controller('AppCtrl', ['$scope', '$rootScope', 'localStorageService',
'stockTicker', function($scope, $rootScope, localStorageService, sharedOrders) {
  $scope.welcome = "Angular JS Local storage Stock Ticker";
  $scope.mystock = localStorageService.get('localStock');
  $scope.source = "Local";
if ($scope.mystock == null) {
stockTicker.getStock().then(function(response) {
      $scope.mystock = response;
      $rootScope.$broadcast('stockInfo',response);
localStorageService.add('localStock',$scope.mystock);
      $scope.source = "Online";
    })
  }
else {
console.log('No local storage for stock info.');
  }
}]);
```

AngularJS is a highly Responsive Web Development framework.

Cross Browser Support with Responsive Web Development AngularJS is a highly Responsive Web Development framework. With directives like ng-class we can provide device specific rendering capabilities using css. With directives like rwd image map, we can also provide device friendly image rendering. With the help of resolution specific styles we shall dictate different display settings using @media CSS rule.

Its syntax is as follows

```
@media <media-query> {
  /* media-specific rules */
}
media_query_list: <media_query> [, <media_query> ]*
media_query: [[only | not]? <media_type>[ and<expression> ]*]
  | <expression>[ and<expression> ]*
expression: ( <media_feature> [: <value>]? )
media_type: all | aural | braille | handheld | print |
projection | screen | tty | tv | embossed
media_feature: width | min-width | max-width
  | height | min-height | max-height
  | device-width | min-device-width | max-device-width
  | device-height | min-device-height | max-device-height
  | aspect-ratio | min-aspect-ratio | max-aspect-ratio
  | device-aspect-ratio | min-device-aspect-ratio | max-device-aspect-ratio
  | color | min-color | max-color
  | color-index | min-color-index | max-color-index
  | monochrome | min-monochrome | max-monochrome
  | resolution | min-resolution | max-resolution
  | scan | grid

Examples
@media (max-width: 1200px), handheld and (orientation: landscape) { ... }
@media (max-width: 700px) { ... }
```

Steps for web analytics using Google's library and its service

AngularJS for Analytics

User behaviour and usage pattern can be analysed in client end with help of various plugins available for Angular JS.

When we develop a web application and want to measure the usage patterns, we need to include any of the analytics libraries to our code. This enables our application to send the usage/event data to the common analytic provider. From the provider we shall get valuable metrics about our web application. Following are the steps for web analytics using Google's library and its service.

- When we need to perform web analytics, we need to sign up with Google analytics.
  - In this we will give the details of our web application and its deployment details.
- Then we should configure the application parameters for tracking.
  - This includes attributes like the
    - User information
    - Element being tracked
    - Linking key words or Ad words in the web page with our measuring parameters
    - Data importing parameters.
  - We can also drop data from known/testing users and the handle the rest
- We can track the elements based on the following parameters
  - Target URL
    - With anchor tag we can add directives to track the usage of this
  - Time factors
    - When any media is played, its duration can be tracked
  - Views per visit
    - Web pages / any fragment of the web viewed per user session/visit can be tracked
  - Any event that has been performed
    - Button / anchor click etc., can be tracked

Following are code snippets with Angulartics, a module of AngularJS that performs web analytics based on Google analytics plugin.

Following is the tag attribute directive to bind analytics for anchor tag

```
<a href="#" ga="[['set', 'metric1', 10], ['send', 'event', 'player', 'play', video.id]]"></a>
Following is the tracking code snippet that can be used in controller
app.run(function ($rootScope, $location, ga) {
    $rootScope.$on('$routeChangeSuccess', function(){
ga('send', 'pageview');
    });
});
```

- Reports

Once we have the data available based on the configurations we have made we can provide reports. This can be segmented based on geographical location, user group, medium of access, referrer source etc.

AngularJS for Dashboard

AngularJS has got rich chart plug-ins, local support for analytical processing and AJAX/web socket based data access. With this we shall bring in dynamic dashboard being rendered at the client end when server pushes data.

Following code fragment gives outline on client side Web Socket processing to receive and bind data

```
<!doctype html>
<html ng-app="dashboard">
<head>
<scriptsrc="http://d3js.org/d3.v2.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.8/angular.js"></script>
<script>
angular.module('components', [])
   .factory('StockTicker', function() {
varurl = "ws:///localhost:8080/Dashboard/stockTicker";
return {
init : function() {
var self = this;
this.connection = new WebSocket(url);
this.connection.onopen = function() {
console.log("connected");
self.updatevalues();
            }
this.connection.onclose = function() {
console.log("onclose");
            }
this.connection.onerror = function (error) {
console.log(error);
            }
this.connection.onmessage = function(event) {
console.log("refresh");
var data = angular.fromJson(event.data);
self.refresh(data.result);
            }
        },
updatevalues : function(id, value) {
this.sendMessage({
method : "updatevalues",
params : {
id : id,
                    //logic here to append the new value to the existing array of data
                }
            });
        }
    }
  });
functionMainCtrl($scope, StockTicker) {
   $scope.updatevalues = function(element, value) {
var id = element.attr("id");
StockTicker.updatevalues(id, value);
   }
StockTicker.init();
      //chart consteuction logic and binding it to visualization tag here...
}
</script>
</head>
<body ng-controller="MainCtrl">
<gh-visualization val="data" grouped="grouped"></gh-visualization>
</body>
</html>
```

HighCharts, NvD3,chartJS and xCharts are few extensions

AngularJS for Charts/Visualization

AngularJS has plenty of charting plug-in libraries. HighCharts, NvD3,chartJS and xCharts are few extension over angular for visualization.

Charting Plug-ins HighCharts, NvD3,chartJS and xCharts are few extension over angular for visualization.

Angular High Charts Directive This is an angular directive that wraps libraries from highchart to build various charts, and allows easily and modular integration of angular and highcharts. Highcharts currently supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange, bubble, box plot, error bars, funnel, waterfall and polar chart types.

Angular NVD3 This provides a wrapper for d3 visualization library. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Angular ChartJS This is a directive for chartJS library. ChartJS is a HTML5 Compliant visualization library. This is a robust and simple ,robust and interactive charting library.

Angular-xcharts ng-xCharts is a directive that allows you to add xCharts to your angular app. xChart is a framework over D3 charts to provide custom charts with HTML ,CSS and SVG.

AngularJS for Multitenancy

AngularJS is well suited to provide UI based multi-tenancy. Here, we have same data model which can be presented differently for various tenants.

Following example demonstrates same data model being represented with different UI widgets namely, accordion and tab.

```
<!doctype html>
<html ng-app="plunker">
<head>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.8/angular.js"></script>
<script src="http://angular-ui.github.io/bootstrap/ui-bootstrap-tpls-0.10.0.js"></script>
<script>
var app = angular.module('plunker', ['ui.bootstrap']);
app.controller('MainController',
function($scope) {
      $scope.items = [{
name: "item1",
desc: "Item 1",
subitems: [{
name: "subitem11",
desc: "Sub-Item 11"
        }, {
name: "subitem12",
desc: "Sub-Item 12"
        }, {
name: "subitem13",
desc: "Sub-Item 13"
        }]
      }, {
name: "item2",
desc: "Item 2",
subitems: [{
name: "subitem21",
desc: "Sub-Item 21"
        }, {
name: "subitem22",
desc: "Sub-Item 22"
        }, {
name: "subitem23",
desc: "Sub-Item 23"
        }]
      }, {
name: "item3",
desc: "Item 3",
subitems: [{
name: "subitem31",
desc: "Sub-Item 31"
        }, {
name: "subitem32",
desc: "Sub-Item 32"
        }, {
name: "subitem33",
desc: "Sub-Item 33"
        }]
      }];
$scope.
default = $scope.items[2];
    });
app.controller('ItemController', ['$scope',
function(scope) {
      scope.$parent.isopen = (scope.$parent.
default === scope.item);
scope.$watch('isopen', function(newvalue, oldvalue, scope) {
        scope.$parent.isopen = newvalue;
      });
    }
  ]);
</script>
<link href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="mycontainer" ng-controller="MainController">
<div class="left-nav">
<accordion close-others="true">
<accordion-group is-open="isopen" ng-repeat="item in items" ng-
controller="ItemController">
<accordion-heading>
          {{item.desc}} - {{isopen}}
<I class="pull-right glyphicon" ng-class="{'glyphicon-chevron-down': isopen, 'glyphicon-
chevron-right': !isopen}"></i>
</accordion-heading>
<div ng-repeat="subitem in item.subitems">{{subitem.desc}}</div>
</accordion-group>
</accordion>
</div>
<div class="content">
</div>
</div>
<div class="navbarnavbar-inverse" role="navigation" ng-controller="MainController">
<tabset>
<tab  ng-repeat="item in items"  heading="{{item.desc}}" active="{{items.active}}" >
<div>
<div ng-repeat="subitem in item.subitems">{{subitem.desc}}</div>
<I> {{item.desc}}</i>
</div>
</tab>
</tabset>
</div>
</body>
</html>
```

AngularJS supports two way data binding between model and view

AngularJS for Performance

AngularJS can be opted for high performing web applications. This provides high performance in three categories.

- It reduces load to the server
  - With AngularJS we can have local storage. This directly reduces load to the server.
  - Also validations are done at client itself, which in-turn reduces server hits.
  - With AJAX calls we just bring in data to the client, rather than the entire content is being cooked up at server end.
  - AngularJS is a very thin sized robust framework within which any other JavaScript library can be used. This makes it faster in getting loaded and rendering UI.
- Rich UI rendering
  - AngularJS does not need any object modelling for UI rendering. Only custom event handlers and data are being modelled.
  - Many other frameworks need classes to be created to craft UI or at least any other equivalent XML based approach to render UI which is time consuming.
  - This exploits most of the HTML5 features to lessen the custom coding to make any rendering UI easier
- Event Handling
  - Data model are stored locally in $scope which can be very easily pulled across any event handlers. This makes event handlers to react faster.
  - AngularJS supports two way data binding between model and view.
  - Many validations are available as a part the component itself as AngularJS adheres to HTML5.
  - Two-way binding and auto-validation remove most of the custom event handling to update view and/or model.

Overview of AngularJS Application Architecture

AngularJS utilizes 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-View Model) architecture.

Angular JS utilizes the basic principles behind the original MVC software design pattern into how it builds client-side web applications

Model Synchronization with Server Models will be consumed directly either using SOAP or REST services from Angular JS web application. To generate REST calls and process its response we need XML and JSON libraries at the client end. This is available as a part of java script language inside script tag. For handling SOAP services, to cook up requests and to parse down responses many custom ready to use code / libraries are available.

AngularJS – Consuming SOAP Services using JS Code

```
<html>
<head>
<title>Angular SOAP request using javascript</title>
<script>
function soapReq() {
var xmlhttp = new XMLHttpRequest();
xmlhttp.open('POST', 'https://myserver.com/', true);
        // build SOAP request
var sr =
            '<?xml version="1.0" encoding="utf-8"?>' +
            '<soapenv:Envelope ' +
              'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ' +
              'xmlns:my="http://myserver.com/mycontext/" ' +
              'xmlns:xsd="http://www.w3.org/2001/XMLSchema" ' +
              'xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">' +
              '<soapenv:Body>' +
                '<api:authenticate
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">' +
            '<api:username xsi:type="xsd:string">login_username</api:username>' +
              '<api:password xsi:type="xsd:string">password</api:password>' +
                '</api:authenticate>' +
              '</soapenv:Body>' +
            '</soapenv:Envelope>';
        xmlhttp.onreadystatechange = function () {
if (xmlhttp.readyState == 4) {
if (xmlhttp.status == 200) {
                //process result in case of asynchronous request
              }
            }
          }
        // Send the POST request
xmlhttp.setRequestHeader('Content-Type', 'text/xml');
xmlhttp.send(sr);
        // process the result here if synchronous request
        // ...
      }
</script>
</head>
<body>
<form name="Demo" action="" method="post">
<div>
<input type="button" value="Soap" onclick="soapReq();" />
</div>
</form>
</body>
<html>
```

Widgets and UI modules support to create a seamless user experience for all types of RCA

AngularJS – Consuming SOAP Services using Libraries Libraries are available to post SOAP request from javascript. Generally such libraries will perform following functionalities

- ➜ It gets the WSDL and caches the description for future requests
- ➜ It prepares and sends a SOAP request to the server (invoking method and parameter values)
- ➜ It processes the server reply using the WSDL so as to build the corresponding JavaScript objects to be returned
- ➜ If the call mode is async, the CallBack method is invoked, otherwise it returns the corresponding object

We can find one such javascript library from http://javascriptsoapclient.codeplex.com/. Following is a simple example to make SOAP request using SOAPClient library

```
<html>
<head>
<script language="JavaScript" type="text/javascript" src="soapclient.js"></script>
<script type="text/javascript">
function login() {
var url2 = "https://myserver.com/";
var pl = new SOAPClientParameters();
pl.add("username", login_username);
pl.add("password", password);
SOAPClient.invoke(url2, "methodToCall", pl, true, getDataCallback);
      }
function getDataCallback(r, soapResponse) {
alert(r.contents.authenticatedResult);
      }
</script>
</head>
<body>
<form>
<input type="button" value="Click Here to login" onClick="getData()" style="width:
192px">
</form>
<div id="result">Result?</div>
</body>
</html>
```

AngularJS Eco System

As AngularJS offers a variety of ways in creating sophisticated RCA, it also has the great eco system with largest widgets and UI modules to offer rich user experience. The following list captures its widgets and UI modules support to create a seamless user experience for all types of RCA.

Widgetsand Event Handlers AngularJS provides a wrapper over basic html input elements to support two way data binding. The following table shows the its usage.

Widget	Tag
CheckBox	☐
DateTimeLocal	<input type="dateTimeLocal"/>
Date	<input type="date"/>
Email	<input type="email"/>
Month	<input type="month"/>
Numbers	<input type="number" />
Radio button	☐
Text	____
Time	<input type="time"/>
URL	<input type="url"/>
Week	<input type="week"/>
List box	<select/>
Text area	<textarea/>

UI Modules External Third-party Rich UI widget collections comes as a great options for Angular JS, following are few examples of rich client widgets that can be easily integrated with any Angular JS code.

Widget	Attribute/Tag directive	JavaScript URL
Accordian	Accordion/accordion group tag	https://github.com/angular-ui/bootstrap/blob/gh-pages/ui-bootstrap-0.1.0-SNAPSHOT.min.js
Alerts&dialog boxes	Div tag and controllers	
Button	Button and div tags	
Carousel	Carousel tag	
Datepicker, Time Picker	Datepicker and timepicker tag	
Drop down	Dropdown directive	
Pagination widgets	Pagination tag	
Popover and tooltip	Popover and tooltip directive	
Progressbar	Progressbar tag	
Rating	Rating tag	
Tabs	Tab, Tab set, Tab heading tags	
Grid	Ng-grid directive	http://angular-ui.github.io/ng-grid/lib/ng-grid.js
File Upload	ng-file-select directive	http://angular-file-upload.appspot.com/js/angular-file-upload.js
Split pane	Splitter directive	http://ngmodules.org/modules/bg-splitter

Integrating third-party UI module plug-in is much easier in Angular JS with the following steps

- *Start linking to the appropriate CSS files.*
- *Include java script file.*
- *Use the tag or attribute directive.*

See the Example below

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<meta charset=utf-8 />
<title>bg-splitter</title>
<link rel="stylesheet" href="css/style.css" media="screen" type="text/css" />
</head>
<body>
<bg-splitter orientation="horizontal">
<bg-pane min-size="100" class="pane-container"><div>left Pane</div></bg-pane>
<bg-pane min-size="150">
<bg-splitter orientation="vertical">
<bg-pane min-size="100" class="pane-container"><div>top right Pane</div></bg-
pane>
<bg-pane min-size="150" class="pane-container"><div>bottom right
Pane<div></bg-pane>
</bg-splitter>
</bg-pane>
</bg-splitter>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.3/angular.min.js"></script>
<scriptsrc="js/splitter.js"></script>
<script>
angular.module('myApp', ['bgDirectives']);
</script>
</body>
</html>
```

AngularJS inherently handles following two major security threats

Application Security

AngularJS supports two forms of security. One is cookie based security. This is done by providing credentials to the server and getting a cookie to have further requests authenticated. Another is Token based, in which a token is shared as part of all requests that is raised which can be validated at the server end. AngularJS inherently handles following two major security threats

JSON Vulnerability Protection: A JSON vulnerability allows third party website to turn your JSON resource URL into JSONP request under some conditions. To counter this your server can prefix all JSON requests with following string ")]}',\n". Angular will automatically strip the prefix before processing it as JSON. For example

For example if your server needs to return:

['one','two']

which is vulnerable to attack, your server can return:

)]}',

['one','two']

If any json content is received from server, Angular will check for the existence of)]}' and it will strip this prefix, before processing the JSON.

The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP. All the requests and responses through this $http service are transformed to handle JSON vulnerabilities. Both requests and responses can be transformed using transform functions. By default, Angular applies following transformations.

Request transformations:

- If the data property of the request configuration object contains an object, serialize it into JSON format.

Response transformations:

- If XSRF prefix is detected, strip it (see Security Considerations section below).
- If JSON response is detected, deserialize it using a JSON parser.

Cross Site Request Forgery (XSRF) Protection: XSRF is a technique by which an unauthorized site can gain your user's private data. Angular provides a mechanism to counter XSRF. When performing XHR requests, the $http service reads a token from a cookie (by default, XSRF-TOKEN) and sets it as an HTTP header (X-XSRF-TOKEN). Since only JavaScript that runs on your domain could read the cookie, your server can be assured that the XHR came from JavaScript running on your domain. The header will not be set for cross-domain requests.

Multiple WAR files suits when the server application has to be hosted in a separate environment from the Angular application.

Application Packaging

There are two approaches to packaging application artefacts for deployment. One is bundling Server side REST/SOAP handlers in a separate war and Angular JS based UI components as a separate war. Another approach is that both server side components and UI artifacts can be bundled in a single war itself.

Isolated / Multiple WAR Files This is more suitable when the server application has to be hosted in a separate environment from the Angular application.

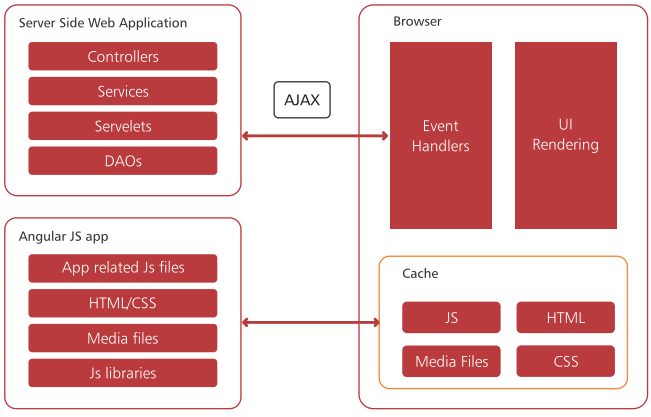
In this approach

- All the files have to be put on to the server in a directory.
- This directory has to be mapped in the web server to be accessed locally.
- To access web service resources like REST or SOAP, we need to refer them as if they are in a different context

Typical Directory structure in this approach

```
app/
css/
img/
js/
      <All JS files goes here in categorized sub-directories>

restapp/
      WEB-INF/
      classes
      <All class files goes here in categorized sub-directories>
                              lib/<containing jar files>
                              web.xml
```

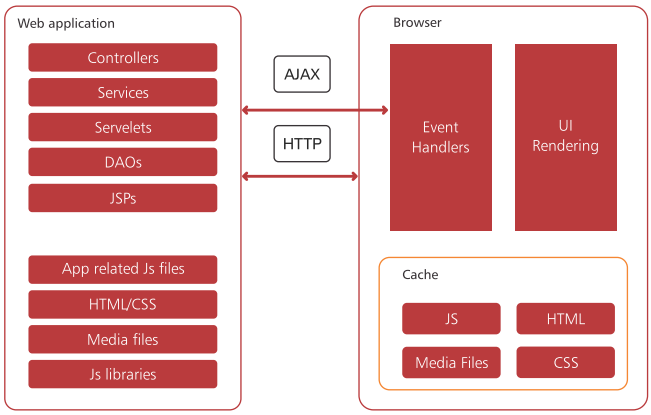


Adopt AngularJS for building sophisticated Interactive Web Application

Single WAR File When application artefacts including both server application and client JS applications are packed inside the war, we should ensure all the file are bundled inside WEB-INF directory to be served automatically. In this case web service URLs can be referred as just local context resources. This is more suitable when server app and UI Angular app are deployed together. This is preferred for architectural decoupling of UI and server components for easy maintainability and extensibility.

Directory structure in this format.

```
combinedapp/
    WEB-INF/
            classes
            <All class files goes here in categorized sub-directories>
            lib/<containing jar files>
            web.xml
            css/
            img/
            js/
            <All JS files goes here in categorized sub-directories>
            lib/
            <thirdparty js files and plugins>
```



Conclusion

The objective of this white paper is to bring the Angular JS capabilities to the forefront in creating various RIA covering from typical SPA to complicated web application incorporating the combination of Charts, Widgets, Dashboard and bringing in Multi-tenant experience. However, this is not a development guide to build application using Angular JS but ushers in the interest to adopt Angular JS for building sophisticated Interactive Web Application.

References

http://andyshora.com/unit-testing-best-practices-angularjs.html

http://www.informit.com/articles/article.aspx?p=1278984

http://code.tutsplus.com/tutorials/5-awesome-angularjs-features--net-25651

https://docs.angularjs.org/tutorial/step_02

http://codeutopia.net/blog/2013/03/16/knockout-vs-backbone-vs-angular/

http://blog.nebithi.com/knockoutjs-vs-angularjs/

http://tutorials.jenkov.com/angularjs/custom-directives.html

https://docs.angularjs.org/guide/

ABOUT ASPIRE

Aspire Systems is a global technology services firm serving as a trusted technology partner for our customers. We work with some of the world's most innovative enterprises and independent software vendors, helping them leverage technology and outsourcing in our specific areas of expertise. Our services include Product Engineering, Enterprise Transformation, Oracle Application Practice, Independent Testing Services and IT Infrastructure Support services. Our core philosophy of "Attention. Always." communicates our belief in lavishing care and attention on our customers and employees.

NORTH AMERICA | UK | BENELUX | NORDIC | MIDDLE EAST | INDIA
+91 - 044 - 67404000, +1- 630- 368 - 0970, +44 - 203 170 6115

For more info contact
info@aspiresys.com or visit www.aspiresys.com