Angular JS Coding Standards

(With Architectural Flow)

VERSION CONTROL

Prepared by:	Baskaran
Date:	6/10/2015
Reviewed and Accepted by:	B.C.Subramanian
Date:	7/10/2015
Approved by:	Baskaran Varadarajan, Soundararajan Das
Date:	28/12/2015

VERSION HISTORY

Date	Version	Author	Description
6/10/2015	0.1	Baskar	Initial Draft Version
9/10/2015	1.0	Bala	Final Reviewed
28/12/2015	1.1	Baskaran	Final Approved

Table of Contents

1. ANGULARJS CODING STANDARDS

1.1 INTRODUCTION

The goal of this style guide is to present a set of best practices and style guidelines for AngularJS applications.

In this style guide you won't find common guidelines for JavaScript development.

2. GENERAL

2.1 Directory structure

Since a large AngularJS application has many components it's best to structure it in a directory hierarchy.

Creating high-level divisions by component types and lower-level divisions by functionality.

In this way the directory structure will look like:

```
├── app
│   ├── app.js
│   ├── controllers
│   │   ├── home
│   │   │   ├── FirstCtrl.js
│   │   │   └── SecondCtrl.js
│   │   └── about
│   │       └── ThirdCtrl.js
│   ├── directives
│   │   ├── home
│   │   │   └── directive1.js
│   │   └── about
│   │       ├── directive2.js
│   │       └── directive3.js
│   ├── filters
│   │   ├── home
│   │   └── about
│   └── services
│       ├── CommonService.js
│       ├── cache
│       │   ├── Cache1.js
│       │   └── Cache2.js
│       └── models
│           ├── Model1.js
```

```
│       └── Model2.js
├── partials
├── lib
└── test
```

2.2 Markup

Put the scripts at the bottom.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyApp</title>
</head>
<body>
  <div ng-app="myApp">
    <div ng-view></div>
  </div>
  <script src="angular.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

2.3 OTHERS

- Use:
  - $timeout instead of setTimeout
  - $interval instead of setInterval
  - $window instead of window
  - $document instead of document
  - $http instead of $.ajax

This will make your testing easier and in some cases prevent unexpected behaviour (for example, if you missed $scope.$apply in setTimeout).

- Automate your workflow using tools like:

- Yeoman
- Grunt
- Bower

- Use promises ($q) instead of callbacks. It will make your code look more elegant and clean, and save you from callback hell.
- Use $resource instead of $http when possible. The higher level of abstraction will save you from redundancy.
- Use an AngularJS pre-minifier (like ngmin or ng-annotate) for preventing problems after minification.
- Don't use globals. Resolve all dependencies using Dependency Injection.
- Do not pollute your $scope. Only add functions and variables that are being used in the templates.
- Prefer the usage of controllers instead of ngInit. The only appropriate use of ngInit is for aliasing special properties of ngRepeat. Besides this case, you should use controllers rather thanngInit to initialize values on a scope.
- Do not use $ prefix for the names of variables, properties and methods. This prefix is reserved for AngularJS usage.

2.4 Optimize the digest cycle

- Watch only the most vital variables (for example: when using real-time communication, don't cause a$digest loop in each received message).
- For content that is initialized only once and then never changed, use single-time watchers likebindonce.
- Make computations in $watch as simple as possible. Making heavy and slow computations in a single $watch will slow down the whole application (the $digest loop is done in a single thread because of the single-threaded nature of JavaScript).
- Set third parameter in $timeout function to false to skip the $digest loop when no watched variables are impacted by the invocation of the $timeout callback function.

3. MODULES

Modules should be named with lowerCamelCase. For indicating that module b is submodule of module a you can nest them by using namespacing like: a.b.

4. CONTROLLERS

- Do not manipulate DOM in your controllers, this will make your controllers harder for testing and will violate the Separation of Concerns principle. Use directives instead.
- The naming of the controller is done using the controller's functionality (for example shopping cart, homepage, admin panel) and the substring Controller in the end. The controllers are named UpperCamelCase (HomePageController, ShoppingCartController, AdminPanelController, etc.).

5. DIRECTIVES

- Name your directives with lowerCamelCase
- Use scope instead of $scope in your link function. In the compile, post/pre link functions you have already defined arguments which will be passed when the function is invoked, you won't be able to change them using DI. This style is also used in AngularJS's source code.
- Use custom prefixes (Power Standings uses the prefix 'ps') for your directives to prevent name collisions with third-party libraries.
- Do not use ng or ui prefixes since they are reserved for AngularJS and AngularJS UI usage.
- DOM manipulations must be done only through directives.
- Create an isolated scope when you develop reusable components.
- Use directives as attributes or elements instead of comments or classes, this will make your code more readable.
- Use $scope.$on('$destroy', fn) for cleaning up. This is especially useful when you're wrapping third-party plugins as directives.
- Do not forget to use $sce when you should deal with untrusted content.
- Services
- Use camelCase (lower) to name your services.
- Encapsulate business logic in services.
- For session-level cache you can use $cacheFactory. This should be used to cache results from requests or heavy computations.

6. TEMPLATES

- Think about using ng-bind or ng-cloak instead of simple {{ }} to prevent flashing content on initial pages of multi view apps.
- Avoid writing complex expressions in the templates.
- When you need to set the src of an image dynamically use ng-src instead of src with {{}}template.
- Instead of using scope variable as string and using it with style attribute with {{ }}, use the directive ng-style with object-like parameters and scope variables as values.

7. ROUTING

- Use resolve to resolve dependencies before the view is shown.

8. SONAR RULES FOR JAVASCRIPT

The following Javascript rules should be used when verifying the Java Script code with Sonar

- Avoid trailing whitespaces
- Sections of code should not be "commented out"
- "alert(...)" should not be used
- Each statement must end with a semicolon
- "===" and "!==" should be used instead of "==" and "!="
- Always use curly braces for "if/else/for/while/do" statements
- Avoid empty block
- "Debugger" statement must not be used
- "TODO" tags should be handled
- Avoid trailing comment
- Declare variables before usage
- Do not use HTML-style comments
- Function names should comply with a naming convention
- Unreachable code
- Unused local variables should be removed
- Unused function parameters should be removed
- Use Javascript "strict" mode with caution

Equivalent Sonar Rule File

JavaScript_Sonar.csv

9. JAVASCRIPT CODING STANDARDS

- Begin names with a character (lower or uppercase as defined below) or underscore, nothing else. Subsequent characters may be any letter, digit or an underscore.
- Use names that are descriptive
- Do not place any spaces in the name
- For consistency in naming variables, use initial capital letters in variable names (do not capitalize prefixes).
- Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
- A block comment should be preceded by a blank line to set it apart from the rest of the code.
- Avoid lines longer than 80 characters.
- All variables should be declared before used. JavaScript does not require this, but doing so makes the program easier to read and makes it easier to detect undeclared variables that may become implied globals. Implied global variables should never be used. Use of global variables should be minimized.
- All functions should be declared before they are used. Inner functions should follow the var statement. This helps make it clear what variables are included in its scope.
- A return statement with a value should not use ( ) (parentheses) around the value. The return value expression must start on the same line as the return keyword in order to avoid semicolon insertion.