NodeJS & AngularJS Coding Standard Guidelines

CONTENTS

1 DOCUMENT DETAILS 4
1.1 Document History 4
1.2 Definition, Acronyms and Abbreviations 4
1.3 References 4
2 INTRODUCTION 5
2.1 Purpose of the Document 5
2.2 Scope of the Document 5
3 GUIDANCE 6
3.1 Naming Convention 6
3.1.1 Classes 6
3.1.2 Variables 6
3.1.3 Functions 6
3.2 File Naming and Organization 6
3.2.1 Header Files 6
3.3 Formatting and Indentation 6
3.4 Comments and Documentation 7
3.5 General Dos and Don'ts 7
3.6 Copyright 7

TABLES

Table 1: Document History... 4
Table 2: Definition, Acronyms and Abbreviations 4
Table 3: References.. 4

1 DOCUMENT DETAILS

1.1 Document History

Version	Author		Reviewer		Approver	
	Name	**Date (DD-MM-YYYY)**	**Name**	**Date (DD-MMM-YYYY)**	**Name**	**Date (DD-MM-YYYY)**
Baseline 1.0	Ravi Vagadia	12-Dec-16	Nirali Shukla	15-Dec-2016	Bijal Chudgar	9-Jan-2017

Version	Description of Change
Baseline 1.0	Created initial version

Table 1: Document History

1.2 Definition, Acronyms and Abbreviations

Definition/Acronym/Abbreviation	Description
N/A	N/A

Table 2: Definition, Acronyms and Abbreviations

1.3 References

No.	Document	Version	Remarks
1.	N/A	-	-

Table 3: References

2 INTRODUCTION

2.1 Purpose of the Document

The purpose of this document is to define the coding guidelines for JavaScript (NodeJS & Angular JS) so that standard and uniform way of coding is followed by all the project team members in the project. This document will serve as a single reference document for all coding activity for JavaScript (NodeJS & Angular JS) projects.

2.2 Scope of the Document

Scope of this document is to provide a clear guidance on adhering to consistent naming conventions, coding style, indentation, anti-patterns, commenting across

3 GUIDANCE

3.1 Naming Convention

- Use full English descriptors that accurately describe the variable/field/class/type.
  - For example, use names like firstName, grandTotal, or CorporateCustomer.

- Use terminology applicable to the domain.
  - e.g: If users of system refer to their clients as Customer, then use term Customer for class, not client or something else.

- Use camelCase to make names readable.
- Use abbreviations sparingly, but if you do so then use then intelligently and document it.
- Avoid long names.
- Avoid names that are similar or differ only in case.

3.1.1 Classes

Classes in ES6 are syntactical sugar over prototype implementation, avoid mixing Class and Prototypes in same code base and avoid modify/accessing class members

3.1.2 Variables

Declare all local variables with either `const` or `let`. Use const by default, unless a variable needs to be reassigned. The `var` keyword must not be used.
One variable per declaration, i.e: avoid using var a=0, b=2;

3.1.3 Functions

Non-Anonymous functions should be declared before use.
Method name should always start with lower case letter and use camelCase.

3.2 File Naming and Organization

Use all lowercase filenames. There are some operating systems that are not case sensitive for filenames and using all lowercase prevents inadvertently using two files that differ only in case that might not work on some operating systems.

Don't use spaces in the filename.
A dash is OK for a word separator. If you want to use some sort of separator for multiple words instead of a space or camelcase as in various-scripts.js, a dash is a safe and useful and commonly used separator.

Think about using version numbers (or a git commit hash) in your filenames (when bundling). When you want to upgrade your scripts, plan for the effects of browser or CDN caching.

3.2.1 Header Files

N/A

3.3 Formatting and Indentation

Use Java style braces structure, along with tabs for indentation.

Use tools like eslint to enforce formatting.

3.4 Comments and Documentation

Use swagger or JSDoc compliant commenting conventions, this is extremely important from collaboration point of view. Please refer and encourage usage of Swagger or JSDoc in your projects.

3.5 General Dos and Don'ts

NodeJS:-

- Install nvm to manage multiple node versions in your development workstation.
- Use package manager (npm or yarn) to manage your project dependencies.
- Identify your project dependencies and use --save or --save-dev with npm properly to identify production dependency or development time dependency.
- Modules should have error-first callback convention. Always check for errors in callbacks. Do not silently ignore error cases, handle it gracefuly and log appropriate message.
- Handle exceptions properly. Use try-catch and promises techniques properly. Avoid nested try-catch and function callbacks.
- Have small modules and exports only those methods that requires public visibility.
- Separate your configuration settings into a config file and use different settings for Production, Development and local environments.
- Make sure to use environment variables to decouple your code to behave as per different environments.
- Use process manager (like pm2, forever) to have a manageable interface to deal with node processes.
- Use logging library to increase error visibility. Do make sure to give respect to methods which are meant to deal with the message that you are going to log.
- There are tons of libraries around nodejs. Make sure that you are not re-inventing the wheel.

AngularJS:-

- Install bower to manage frontend dependencies.
- Leverage modules properly.
- Always let users use expressions whenever possible.
- One should prefer using the dash-delimited format (e.g. ng-model for ngModel). While working with an HTML validating tool, one could instead use the data-prefixed version (e.g. data-ng-model for ngModel).
- Prefer using directives via tag name and attributes over comment and class names. Doing so generally makes it easier to determine what directives a given element matches.
- We should only use scope events to communicate across the controllers in the current screen of our Single Page Application. If we need to only share data, then we should look at using Services instead.
- Controllers should not reference DOM but just contain behavior, Directives should have DOM manipulation.
- Services should have logic independent of view.
- Don't fight with HTML just expand it through Directives.
- It is better to have modular folder structure, so that we can create reusable/ distributable components.
- Dependency injection is one of the best attribute of AngularJS framework and we should always use it. It will really help when we need to cover test cases of our application. Provide alias to dependency so that they will not rename during minification process, because in AngularJS dependencies are resolved by name.

3.6 Copyright

N/A