# Angular Style Guide

# 

**General**

**Directory structure**

* Automate your workflow using tools like:
  + [Yeoman](http://yeoman.io/)
  + [Grunt](http://www.google.com/url?q=http%3A%2F%2Fgruntjs.com%2F&sa=D&sntz=1&usg=AFrqEzfhy_19FGZ_lTba0sEEeufi5EPPhg)
  + [Bower](http://www.google.com/url?q=http%3A%2F%2Fbower.io%2F&sa=D&sntz=1&usg=AFrqEzcpTSWwwDVICUwT9u-6T4lo5gHGUw)

This is the typical folder layout that I suggestion: using the **module pattern**( a module for each feature) to organizing your code

**In this way the directory structure will look like**:

app

├── app

│ ├── about

│ │ ├── about.html

│ │ ├── about.js

│ │ └── init.js

│ ├── dashboard

│ │ ├── feature1

│ │ │ ├── feature1.styl

│ │ │ ├── feature1.html

│ │ │ └── feature1.js

│ │ ├── feature2

│ │ │ └── ...

│ │ ├── dashboard.html

│ │ ├── dashboard.styl

│ │ ├── dashboard.js

│ │ └── init.js

│ ├── template

│ │ ├── sidebar

│ │ │ ├── sidebar.html

│ │ │ ├── sidebar.styl

│ │ │ └── sidebar.js

│ │ ├── header

│ │ │ └── ...

│ │ └── footer.html

│ └── app.js

├── assets

│ ├── img

│ │ └── ...

│ └── font

├── bower\_components

│ └── ...

├── common

│ ├── config

│ │ └── route

│ │ ├── configRoute.js

│ │ └── routeProvider.js

│ ├── directives (ams\_components)

│ │ ├──pagination

│ │ │ ├── pagination.html

│ │ │ ├── pagination.styl

│ │ │ └── pagination.js

│ │ └── ...

│ ├── resources

│ ├── metadata

│ │ └── metadata.js

│ ├── security

│ └── services

├── styles

│ ├── stylus

│ │ └── ...

│ ├── stylesheet.styl

│ ├── theme.styl

│ └── variables.styl

├── 404.html

├── login.html

└── index.htm

**The meaning of files and folders are as follows:**

* **node\_modules** - Node.js modules for various Grunt tasks, usually you don’t have to do anything about this folder
* **test –** this contains folders for the files like karma tests
* **src** - source files, development is done here
  + **assets – assets contains static files – fonts, images.**
  + **app -** with app.js and module folders
    - **dashboard**
      * **feature 1 –** feature folder of dashboard module
      * dashboard.styl - styl file only used this module.
      * dashboard.js- optional file for module definition.
      * init.js **-** new file to contain the module routing.
  + **styles** – contains all the STYL/CSS sources.
  + **bower\_components** –this contains folders for the various vendor libraries such as AngularJS, Angular Animate, JQuery and Bootstrap and 3rd party libraries managed via [Bower](http://www.google.com/url?q=http%3A%2F%2Fbower.io%2F&sa=D&sntz=1&usg=AFrqEzcpTSWwwDVICUwT9u-6T4lo5gHGUw),
  + **common** - is a place for reusable parts of the app – both third party and your own.
    - **config**
      * **routes -** route defines for the app
  + .buildignore
  + .htaccess
  + 404.html– this is the 404 error page that will show up, when the user types in a wrong URL or the Angular app couldn’t find the page mentioned in the URL
  + favicon.ico– this is the icon that will show up in the browser tab of your app.
  + login.html – this will be the login page for your app.
  + index.html – this will be the home page for your app.
  + robot.txt– this is the file where you set rules for the search engine robots or crawlers, telling them what pages they can index and which sections of the app should not be indexed.
* index.html - project index with project pages listed
* Gruntfile.js - [Grunt](http://www.google.com/url?q=http%3A%2F%2Fgruntjs.com%2F&sa=D&sntz=1&usg=AFrqEzfhy_19FGZ_lTba0sEEeufi5EPPhg) file with various automation tasks defined in it
* bower.json - Bower dependencies in the project
* package.json - npm packages dependencies
* .yo-rc.json - Yeoman generator configuration file
* .bowerrc - configuration file for Bower
* .editorconfig - [EditorConfig](http://www.google.com/url?q=http%3A%2F%2Feditorconfig.org%2F&sa=D&sntz=1&usg=AFrqEzfknQlosdkilhNlH5QVJbfI06vOOg) configuration file to achieve consistent coding
* .gitattributes - [Git](http://www.google.com/url?q=http%3A%2F%2Fgit-scm.com%2F&sa=D&sntz=1&usg=AFrqEzdDN-WxlNAp0BLVVg0b2-ltMcAo6g) configuration file to force Unix line ending in all text files
* .gitignore - default Git ignore files and folders
* .jshitrc - [JSHint](http://www.google.com/url?q=http%3A%2F%2Fwww.jshint.com%2F&sa=D&sntz=1&usg=AFrqEzeGqTiKioi3nUU18ekpFqGZYuwH8w) configuration

**Checking bugs**

Always check your javascript code for JSHint error and fix it immediately!

Because WebStorm has this feature, so there is no excuse for not using it!

Following this rule will help eliminate a large set of bugs.

**Naming conventions**

When you're starting a new project, please use the following naming conventions:

* **camelCase** for all variables, methods, and functions (never underscore-separated)
* **UpperCamelCase** for all constructor functions and namespace objects
* **UPPERCASE\_UNDERSCORE\_SEPARATED** for constants.

When working with existing project it is important to recognize and follow the same naming pattern. No matter what conventions are followed, all names should be **descriptive**, identifying what the data variable holds or what the function does.

**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).

* 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.
* **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.

**Directives**

* Directives shouldn't be too long, if it's too long then it is handling multiple responsibilities
* Name your directives with **lowerCamelCase**.
* **DOM manipulations must be done only through directives**.
* **Create an isolated scope when you develop reusable components**.
* U**se directives as attributes or elements instead of comments or classes**, this will make your code more readable.

|  |
| --- |
| **JavaScript** |
| .directive('directiveName', **function** (injectables) {  **return** {  restrict: 'ECMA',  template: '<div></div>',  templateUrl: 'directive.html',  replace: false,  priority: 0,  transclude: false,  scope: false,  terminal: false,  require: false,  controller: ['$scope','$element','$attrs','otherInjectables',function($scope, $element, $attrs, otherInjectables) { ... }],  compile: **function** compile(tElement, tAttrs, transclude) {  return {  pre: **function** preLink(scope, iElement, iAttrs, controller) { ... },  post: **function** postLink(scope, iElement, iAttrs, controller) { ... }  }  },  link: **function** postLink(scope, element, attrs) { ... }  };  }); |

# Controllers

* **Do not manipulate DOM in your controllers**, this will make your controllers harder for testing and will violate the [Separation of Concerns principle](https://www.google.com/url?q=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FSeparation_of_concerns&sa=D&sntz=1&usg=AFrqEzf_4SF9U_ZLU6AfpRKicxrL0VDz0w). Use directives instead.
* The naming of the controller is done using the controller's functionality (for example tree menu, dashboard, admin panel) and the substring Ctrl in the end. The controllers are named UpperCamelCase (DashboardCtrl, TreeMenuCtrl, AdminPanelCtrl, etc.).
* **The controllers should not be defined as globals** (even though AngularJS allows this, it is a bad practice to pollute the global namespace).
* Use array syntax for controller definitions:

|  |
| --- |
| **JavaScript** |
| (function(){  'use strict';  angular.module(‘AMSApp’, []);  module.controller('MyCtrl', ['dependency1', 'dependency2', ..., 'dependencyn', function (dependency1, dependency2, ..., dependencyn) {  *//...body*  }]);  })(); |

Using this type of definition avoids problems with minification. You can automatically generate the array definition from the standard one using tools like [ng-annotate](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Folov%2Fng-annotate&sa=D&sntz=1&usg=AFrqEzfDUdu6ICQ-6XWmr3PLsorXF5uMTg) (and grunt task [grunt-ng-annotate](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Fmzgol%2Fgrunt-ng-annotate&sa=D&sntz=1&usg=AFrqEze4tTGGLnezARl7S6DpN4PXDgdmgQ)).

**Templates**

* Use **ng-bind** or ng-cloak instead of simple {{ }} to prevent flashing content.
* **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.
* When you need to set the href of an anchor tag dynamically use ng-href instead of href 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:

|  |
| --- |
| **HTML** |
| <script>  ...  $scope.divStyle **=** {  width**:** 200,  position**:** 'relative'  };  ...  </script>  <div ng-style="divStyle" ng-bind=”data”></div>; |

[hmchung@tma.com.vn](mailto:hmchung@tma.com.vn)