# Suggested Directory Structure for front-end Development (AngularJS)

* *index.html* – *this is the main entry point as a single page application*
* *app.js*– *this will define the main module, its* ***config****, and all its* ***dependency modules***
* **lib** – *all external libraries must have its own directory encapsulating its own js, css, images files etc.* 
  + **jquery**
    - *jquery.min.js*
  + **angular**
    - *angular.js*
    - *angular.min.js*
  + **bootstrap**
    - **js**
      * *bootstrap.js*
    - **styles**
      * *bootstrap.css*
    - **fonts**
    - **images**
* **scripts** – *this will contain all the scripts that is used by main application*
  + *controllers.js*
  + *directives.js*
  + *services.js*
  + *filters.js*
* **styles** – *this contains all the css files being used by this application* 
  + main.css
  + about\_us.css
  + contact\_us.css
* **images** – *this contains all the images*
* **views** – *this contains the partial html and other full page html*
  + **partials**
    - *login\_form.html*
    - *registration\_form.html*
  + *contact\_us.html*
* **modules** – *this will store all other modules*
  + **first\_business***– example of a small module*
    - *first\_business\_*app.js – *this file defines the module and its dependencies*
    - *first\_business\_controllers.js*
    - *first\_business\_services.js*
    - *first\_business\_directives.js*
    - *first\_business\_filters.js*
    - **views** *– used this only if the html file is exclusive to this module*
    - **styles** *– used this only if the css file is exclusive to this module*
  + **other\_business***– example of a huge module*
    - other\_business\_app.js – *this file defines this module and its dependencies*
    - **controllers** 
      * *other\_business\_function1\_controller.js*
      * *other\_business\_function2\_controller.js*
    - **services**
      * *other\_business\_function1\_service.js*
      * *other\_business\_function2\_service.js*
    - **directives**
      * *other\_business\_function1\_directive.js*
      * *other\_business\_function2\_directive.js*
    - **filters**
      * *other\_business\_function1\_filter.js*
      * *other\_business\_function2\_filter.js*
    - **views** *– used this only if the html file is exclusive to this module*
      * *other\_business.html*
    - **styles** *– used this only if the css file is exclusive to this module*

**Filenames and Directories**: must be all lower case and separate words with underscore ‘**\_**’

For AngularJS, please use the following naming convention and guidelines:

*The following guidelines are community-driven set of best practices for AngularJS application development.*

# 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 word ‘Controller’ in the end. The controllers are named **UpperCamelCase** (HomePageController, AboutUsController, AdminPanelController, ReferenceDateController, 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: Using this type of definition avoids problems with minification.

***Don’t use this****:*

*module.controller('HomePageController', function (dependency1, dependency2, ..., dependencyn) {*

*//...body*

*});*

***Instead use this****:*

*module.controller('HomePageController',* ***[****'dependency1', 'dependency2', ..., 'dependencyn', function (dependency1, dependency2, ..., dependencyn) {*

*//...body*

*}****]****);*

* Use the original names of the controller's dependencies. This will help you produce more readable code.

***Don’t use this****:*

*module.controller('HomePageController', ['$scope', function (****s****) {*

*//...body*

*}]);*

***Instead use this****:*

*module.controller('HomePageController', ['$scope', function (****$scope****) {*

*//...body*

*}]);*

* Make the controllers as lean as possible. Abstract commonly used functions into a service.
* Communicate within different controllers using method invocation (possible when a child wants to communicate with its parent) or $emit, $broadcast and $on methods. The emitted and broadcasted messages should be kept to a minimum.
* Make a list of all messages which are passed using $emit, $broadcast and manage it carefully because of name collisions and possible bugs.
* When you need to format data encapsulate the formatting logic into a filter and declare it as dependency:

module.filter('myFormat', function () {

return function () {

//body...

};

});

module.controller(*'HomePageController'*, ['$scope', 'myFormatFilter', function ($scope, myFormatFilter) {

//body...

}]);

* In case of nested controllers use "nested scoping" (the controllerAs syntax):

<div ng-controller="MainController as main">{{ main.title }}

<div ng-controller="**AnotherController as another**">{{ another.title }}

<div ng-controller="**YetAnotherController as yet**">

{{ yet.title }}

</div>

</div>

</div>

# 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 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** (Strict Contextual Escaping services) when you should deal with untrusted content.

# Filters

* Name your filters with **lowerCamelCase**.
* Make your filters as light as possible. They are called often during the $digest loop so creating a slow filter will slow down your app.
* Do a single thing in your filters, keep them coherent. More complex manipulations can be achieved by piping existing filters.

# Services

* Use camelCase to name your services.
  + UpperCamelCase (PascalCase) for naming your services, used as constructor functions i.e.:

*module.controller('MainController', function ($scope, User) {*

*$scope.user = new User('foo', 42);*

*});*

*module.factory('User', function () {*

*return function User(name, age) {*

*this.name = name;*

*this.age = age;*

*};*

*});*

* **lowerCamelCase** for all other services.
* Encapsulate all the business logic in services.
* For session-level cache you can use $cacheFactory. This should be used to cache results from requests or heavy computations.
* If given service requires configuration define the service as provider and configure it in the config callback like:

*angular.module('demo', [])*

*.config(function ($provide) {*

*$provide.provider('sample', function () {*

*var foo = 42;*

*return {*

*setFoo: function (f) {*

*foo = f;*

*},*

*$get: function () {*

*return {*

*foo: foo*

*};*

*}*

*};*

*});*

*});*

*var demo = angular.module('demo');*

*demo.config(function (sampleProvider) {*

*sampleProvider.setFoo(41);*

*});*

# Templates/HTML portion

* 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:

*<script>*

*...*

*$scope.divStyle = {*

*width: 200,*

*position: 'relative'*

*};*

*...*

*</script>*

<div ng-style="divStyle">my beautifully styled div which will work in IE</div>;

# Routing

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

A resolve is a property you can attach to a route in both [ngRoute](https://docs.angularjs.org/api/ngRoute/provider/$routeProvider) and the more robust [UI router](https://github.com/angular-ui/ui-router).

Please see this link for more info:[Using Resolve In AngularJS Routes](http://odetocode.com/blogs/scott/archive/2014/05/20/using-resolve-in-angularjs-routes.aspx)

* Let us use [UI router](https://github.com/angular-ui/ui-router) rather than [ngRoute](https://docs.angularjs.org/api/ngRoute/provider/$routeProvider) because of it flexibility routing and nested views.

## Others

* Use:
  + $timeout instead of setTimeout
  + $interval instead of setInterval
  + $window instead of window
  + $document instead of document
  + $http instead of $.ajax
* Use promises ($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.
* Prefer the usage of [controllers instead of ngInit](https://github.com/angular/angular.js/pull/4366/files). The only appropriate use of ngInit is for aliasing special properties of ngRepeat. Besides this case, you should use controllers rather than ngInit 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.
* When resolving dependencies through the DI mechanism of AngularJS, sort the dependencies by their type - the built-in AngularJS dependencies should be first, followed by your custom ones:

*module.factory('Service', function ($rootScope, $timeout, MyCustomDependency1, MyCustomDependency2) {*

*return {*

*//Something*

*};*

*});*