Introduction

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

General

Directory structure

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

Root folder

app

css

images

lib

js

search

controller.js

services.js

apps

controller.js

services.js

logvault

controller.js

services.js

global.js

filter.js

directives.js

controller.js
services.js
app.js
partial
search.html
seactionview.html
logvault.html
dashboard.html
index.html

- The app.js file contains route definitions, configuration and/or manual bootstrap
- Each JavaScript file should only hold a single component. The file should be named with the component's name.
- Use Angular project structure template like Yeoman, ng-boilerplate.

Markup

Keep things simple and put AngularJS specific directives later. This way is easy to look to the code and find enhanced HTML by the framework (what improve the maintainibility).

```
<form class="frm" ng-submit="login.authenticate()">
  <div>
    <input class="ipt" type="text" placeholder="name" require ng-model="user.name">
  </div>
</form>
```

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).

- 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.

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.

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 Ctrl in the end. The controllers are named UpperCamelCase (HomePageCtrl, ShoppingCartCtrl, 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:

```
module.controller('MyCtrl', ['dependency1', 'dependency2', ..., 'dependencyn', function
(dependency1, dependency2, ..., dependencyn) {
  //...body
}]);
```

Using this type of definition avoids problems with minification.

```
module.controller('MyCtrl', ['$scope', function (s) {
  //...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.

- 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('MyCtrl', ['$scope', 'myFormatFilter', function ($scope, myFormatFilter) {
  //body...
}]);
```

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.

- 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.

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

- 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.

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.

The diagram below depicts how each component integrates with other components. Here are the steps that occur for the Angular application to run.

1. Web page loads
2. Web page loads the application module
3. The module is configured with routes
4. The routes execute and loads the View and the Controller
5. Services are created and injected into the Controller
6. The view is added to the web page
7. The scope is bound to the view and the Controller