

Best Practices Document

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | |  |
| Version | Date | Author/Contributor | Comments |
| 0.1 | 24-12-2015 | Indra Preet Singh | Compiled first draft. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Below is the style guide for syntax ,conventions and structuring Angular Applications which provides better guidance to the development experience .

1. Avoid jquery with Angular . This is because both follows different style of developing an application . jQuery is more concerned about DOM manipulation whereas Angular is more concerned about architectural decisions .
2. Avoid declaring too many watchers

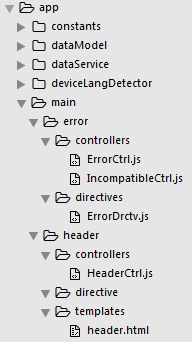
* Keep the count of watchers below 2000. Because of the dirty checking done in a digest cycle, once the number of watchers exceeds about 2,000 the cycle can cause noticeable performance issues.

1. Use Batarang browser plugin for developing and debugging Angular application
2. Keep the directory structured according to the modules and components defined in the application.

The Angular App structure should be modularized into very specific functions . Using the below structure becomes harder to read and understand since the structure is not modularized .



In the below structure the structure is broken down in to specific functionalities. Example , main folder contain the main application components . Thus under main we have different sub components .The error component contains the controllers directives and similarly the header component contains the controllers, directives , templates .



1. Controllers should be kept simple by avoiding data access in controllers . They should defer data calls to a data service / factory . Thus controller should not be concerned with how to retrieve the data . It should just have the data anyhow to allow the view to play with it .

**Example** –

/\* avoid \*/

function OrderController($http, $q, config, userInfo) {

var vm = this;

vm.checkCredit = checkCredit;

vm.isCreditOk;

vm.total = 0;

function checkCredit() {

var settings = {};

return $http.get(settings)

.then(function(data) {

vm.isCreditOk = vm.total <= maxRemainingAmount

})

.catch(function(error) {

});

};

}

/\* recommended \*/

function OrderController(creditService) {

var vm = this;

vm.checkCredit = checkCredit;

vm.isCreditOk;

vm.total = 0;

function checkCredit() {

// Here the data access call has been delegated to the creditService

return creditService.isOrderTotalOk(vm.total)

.then(function(isOk) { vm.isCreditOk = isOk; })

.catch(showError);

};

}

1. Efficient usage of controllerAs syntax

* controllerAs is a syntactic sugar over $scope . You can still bind view and still access $scope methods .

/\* avoid \*/

function CustomerController($scope) {

$scope.name = {};

$scope.sendMessage = function() { };

}

/\* recommended \*/

function CustomerController() {

var vm = this;

vm.name = {};

vm.sendMessage = function() { };

}

* It promotes the use of binding to a "dotted" object in the View (e.g. customer.name instead of name), which is more contextual, easier to read, and avoids any reference issues that may occur without "dotting"

<!-- avoid -->

<div ng-controller="CustomerController">

{{ name }}

</div>

<!-- recommended -->

<div ng-controller="CustomerController as customer">

{{ customer.name }}

</div>

* Use a capture variable for this when using the controllerAs syntax . Choose a consistent variable name such as vm, which stands for ViewModel. The this keyword is contextual and when used within a function inside a controller may change its context. Capturing the context of this avoids encountering this problem.

/\* avoid \*/

function CustomerController() {

this.name = {};

this.sendMessage = function() { };

}

/\* recommended \*/

function CustomerController() {

var vm = this;

vm.name = {};

vm.sendMessage = function() { };

}

1. Business logic belongs to model, services . This business logic should not be a part of controller. For example if we have a **Movie** model. Than which movie has what rating on IMDB should not be the job of controller to decide. It should be the **Movie** model’s job to decide this feature.
2. Declare ‘controller as ’ syntax either in routing or in the view layer . Do not duplicate the usage of ‘controller as ’ syntax .
3. Declare controllers separately for better debugging

Chained style of declaration ( bad  style ) -

*angular.module(‘Sample’).controller(‘AV’,function(){*

*var vm=this;*

*vm.name=’Stark’;*

*})*

Prefer below style –

*angular.module(‘Sample’).controller(‘AV’,AvengerDetails)*

*function AvengerDetails (){*

*var vm=this;*

*vm.name=’Stark’;*

*}*

1. Use IFFE (Immediately Invoked Function Expression ) to  avoid Global Variable pollution.
2. Declare dependencies in a safe minified manner as demoed below -

*angular.module(‘Sample’).controller(‘AV’,AvengerDetails)*

*AvengerDetails.$inject = [‘$q’,’dataservice’];*

*function AvengerDetails ($q,dataservice){*

*var vm=this;*

*vm.name=’Stark’;*

*}*

1. Place bindable members at the top of the controller.

Setting anonymous functions in-line can be easy, but when those functions are more than 1 line of code they can reduce the readability. Defining the functions below the bindable members (the functions will be hoisted) moves the implementation details down, keeps the bindable members up top, and makes it easier to read.

/\* avoid \*/

function SessionsController() {

var vm = this;

vm.gotoSession = function() {

/\* ... \*/

};

vm.refresh = function() {

/\* ... \*/

};

vm.search = function() {

/\* ... \*/

};

vm.sessions = [];

vm.title = 'Sessions';

}

/\* recommended \*/

function SessionsController() {

var vm = this;

vm.gotoSession = gotoSession;

vm.refresh = refresh;

vm.search = search;

vm.sessions = [];

vm.title = 'Sessions';

////////////

function gotoSession() {

/\* \*/

}

function refresh() {

/\* \*/

}

function search() {

/\* \*/

}

}

1. Use Constants across the application .
2. Return a Promise from Data Calls using $http service .
3. Try to limit 1 directive per file .
4. Manipulate DOM in a Directive .

Since DOM manipulation is difficult to test , debug so this manipulation is delegated to directives or can also be done via CSS manipulation to set styles and Angular templates can also be utilized .

1. Handle and log all routing errors using $routeChangeError