Separation of Concerns

John Papa
http://johnpapa.net
Twitter: @john_papa

pluralsight
hardcore developer training

Stakeholder Requests a New Feature

How much existing code will have to change?

How quickly can it be done?

How risky are the changes?

Benefits of Separation of Concerns

- Easier to Maintain
- Easier to Extend
- Reuse More Code
- Easier to Test
- Easier to Isolate Bugs

Separation of Concerns (SoC) helps manage complexity

AngularJS and Separation of Concerns

Aka: Ravioli Code

When your component does everything,
it does nothing well.

Opens Dialogs

Building Models

Pagination Logic

Controller Handles Logic for a View

Get Data via XHR Call

Broadcast Messages

Logging

1

A Component Has One Role

1

One Component Per File

1

Singular Purpose

All Are Important

1

A Component Has One Role

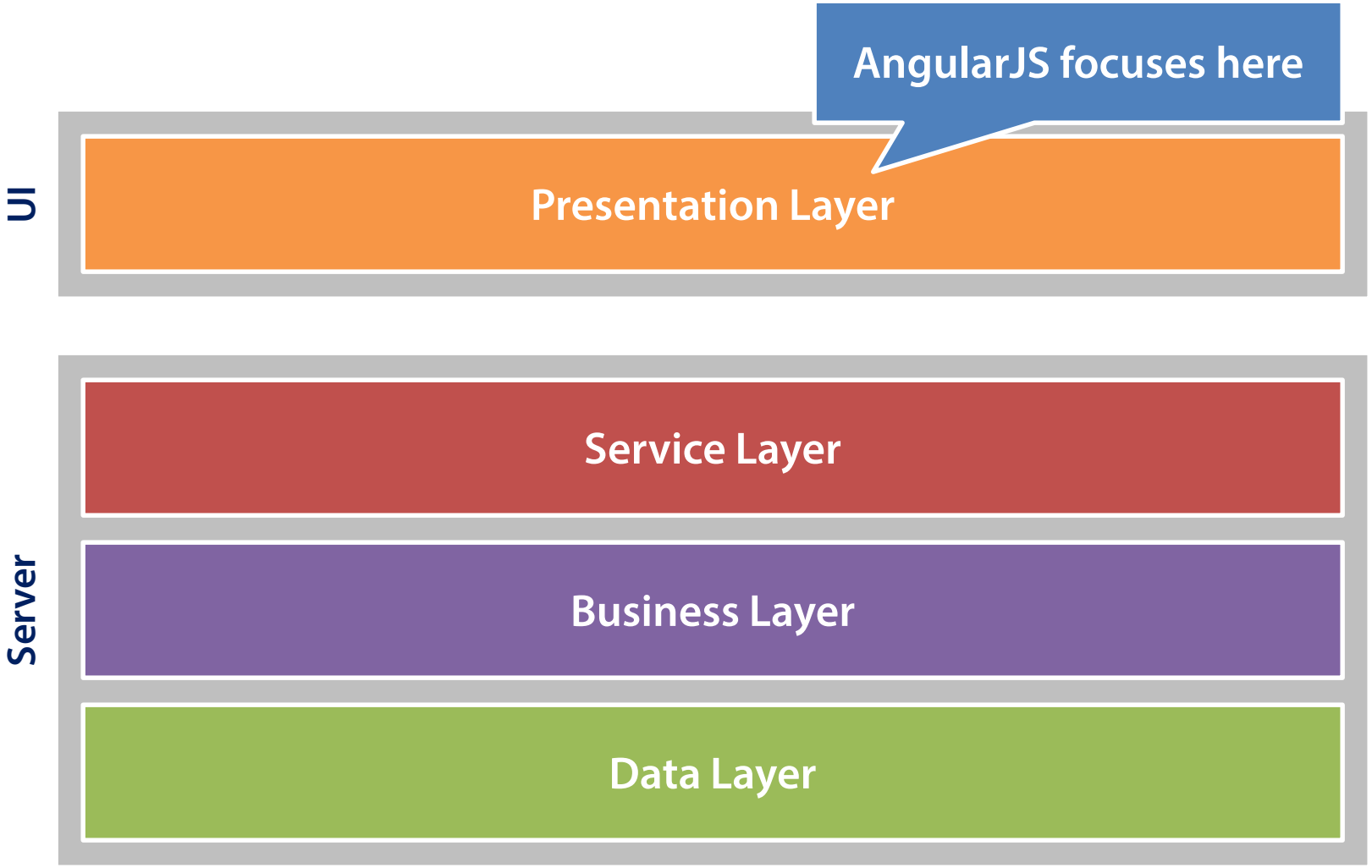
One Component Per File

Singular Purpose

How Do We Separate?

Does Your Component Have 1 Job?

Horizontal SoC



How Do We Think About SoC with AngularJS ?

AngularJS has Familiar Terms

Widgets

Modules

Factories / Services

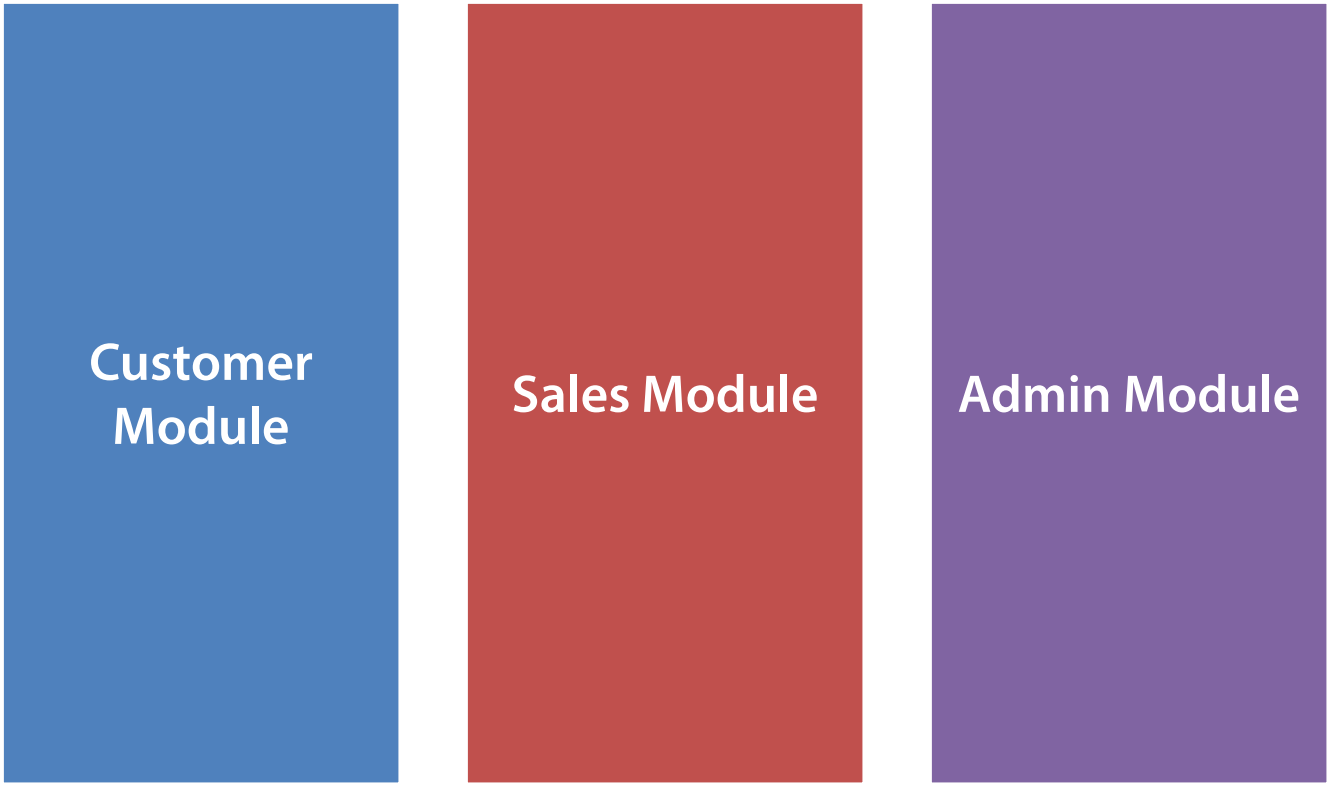
Directives

Models

Views

Controllers

Vertical SoC with Modularity



Vertical SoC within a Module

SoC Examples

Cross cutting and specific to the app

Managing XHR calls to send/receive data (i.e. a data service)

Cross cutting and generic

Logging service

Exception Handling consolidation

Feature and Role

Customer controller

Customer address widget (via a directive)

Easier to Start with Separation

Rule of 1

1

Refactoring Opportunities

```
function avengers($http, $log) {
    var vm = this;
    vm.avengers = [];

    function getAvengers() {
        return $http.get('/api/maa')
            .then(function(data, status, headers, config) {
                vm.avengers = data.data[0].data.results;
                return vm.avengers;
            }, function(error){
                toastr.error(error, title);
                $log.error('Error: ' + error);
```

Other controllers may need to make this call

How we log and show toasts should be reusable

Small, Readable, and Singularly Focused

```
function avengers(avengersData) {
    var vm = this;
    vm.avengers = [];

    function getAvengers() {
        avengersData.getAvengers().then(function (data) {
                vm.avengers = data;
                return vm.avengers;
            });
    }
```

This Controller delegates the work to a dependency

Controllers Defer to Factories

```
angular.module('modularApp.avengers')
    .factory('avengers.dataservice', ['$http', 'common', dataservice]);

function dataservice($http, common) {
    var service = {
        getAvengersCast: getAvengersCast,
        getAvengerCount: getAvengerCount,
        getAvengers: getAvengers
    };

    return service;

    // Implementation details below
```

Value Proposition

Lack of duplication makes it easier to maintain

Increase stability

Easier to extend and enhance

More reuse by other components

Applies to Simple Logic Too

1

Module Startup Logic is Difficult to Test

```
app.run( ['$http', function ($http) {
    $http.get('api/lookupslists');
    $http.get('api/userprofile');
    $rootScope.broadcast('readymessage');
    $log('app is primed');
}]);
```

Inject and Invoke Module Startup Code

```
app.run( ['appStart', function ( appStart ) {
    appStart.start();
}]);
```

Take Advantage of Modularity

1

Group Features into Modules

```
angular.module('modularApp', [
    'ngAnimate',
    'ngRoute',
    'common',
    'modularApp.avengers',
    'modularApp.dashboard',
    'modularApp.layout',
    'modularApp.widgets'
]);
```

Summary

1 Component, 1 Role, 1 File

Use Dependencies

Modularize

Decreases risk, cost, time to deliver

Increases testability, code reuse, maintainability