Buenas Prácticas de AngularJS
Parte II - Controllers

controllerAs View Syntax

Use la sintaxis controllerAs en lugar de la sintaxis clásica con $scope en las vistas.

Por qué?: Promueve el uso de binding con objetos "dotted" en las vistas (ej: customer.name en lugar de name), lo cual es más contextual, fácil de leer y evita problemas de referencias que podríam ocurrir si no se utilizara "dotting".

Por qué?: Ayuda a evitar tener que llamar a $parent en vistas con controladores anidados.

controllerAs View Syntax

```
<!-- avoid -->
<div ng-controller="Customer">
   {{ name }}
</div>

<!-- recommended -->
<div ng-controller="Customer as customer">
  {{ customer.name }}
</div>
```

controllerAs Controller Syntax

Use la sintaxis controllerAs en lugar de la sintaxis clásica con $scope.

La sintaxis controllerAs usa this en los controladores y estos valores son agregados al objeto $scope

Por qué?: controllerAs es un agregaro a la sintaxis clásica con $scope. Se puede seguir bindeando valores en la vista y se puede acceder a métodos dentro de $scope.

Por qué?: Evita la tentación de tener que usar $scope en controladores, ya que el mismo sericio debería ser utilizado dentro de factories o servicios.

controllerAs Controller Syntax

```
/* avoid */
function Customer($scope) {
    $scope.name = {};
    $scope.sendMessage = function() { };
}

/* recommended - but see next section */
function Customer() {
    this.name = {};
    this.sendMessage = function() { };
}
```

controllerAs with vm

Use una variable para capitar la referencia a this cuando use la sintaxis controllerAs. Elija una variable consistente como puede ser vm, que hace referencia a ViewModel.

Por qué?: La palabra clave this es contextual y cuando es usado en una función dentro de un controlador puede ser que su contexto cambie. Capturando el contexto dentro de una variable nos ayuda a evitar este problema.

```
/* avoid */
function Customer() {
    this.name = {};
    this.sendMessage = function() { };
}
/* recommended */
function Customer() {
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}
```

Bindable Members Up Top

Ubique los miembros bindable al principio de nuestro controlador, ordenados alfabéticamente y no esparcidos a lo largo de todo el código del controlador.

Por qué?: Ubicando miembros bindables al principio hacen la lectura del controlador más sencilla y nos ayuda a identificar rápidamente los miembros de nuestro controlador que serán mostrados en la vista.

Por qué?: Setear funciones anónima inline puede ser fácil, pero cuando esas funciones son más extensas de una línea de código, se reduce la legibilidad drásticamente. Definiendo las funciones debajo de los miembros bindables mueve la implemetnación al final ocultando los detalles y hacienda la lectura mucho más sencilla.

Bindable Members Up Top

```
/* avoid */
function Sessions() {
    var vm = this;

    vm.gotoSession = function() {
      /* ... */
    };
    vm.refresh = function() {
      /* ... */
    };
    vm.search = function() {
      /* ... */
    };
    vm.sessions = [];
    vm.title = 'Sessions';
```

```
/* recommended */
function Sessions() {
    var vm = this;

    vm.gotoSession = gotoSession;
    vm.refresh = refresh;
    vm.search = search;
    vm.sessions = [];
    vm.title = 'Sessions';

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

    function gotoSession() {
      /* */
    }

    function refresh() {
      /* */
    }

    function search() {
      /* */
    }
```

Bindable Members Up Top

```
/* avoid */
function Sessions(data) {
    var vm = this;

    vm.gotoSession = gotoSession;
    vm.refresh = function() {
        /**
         * lines
         * of
         * code
         * affects
         * readability
         */
    };
    vm.search = search;
    vm.sessions = [];
    vm.title = 'Sessions';
```

```
/* recommended */
function Sessions(dataservice) {
    var vm = this;

    vm.gotoSession = gotoSession;
    vm.refresh = dataservice.refresh; // 1 liner is OK
    vm.search = search;
    vm.sessions = [];
    vm.title = 'Sessions';
```

Function Declarations to Hide Implementation Details

Use declaración de funciones para oculatar detalles de implementación. Mantenga sus miembros bindables al principio. Cuando necesite exponer una función que será invocada desde la vista, cree un puntero a una función que será luego definida en nuestro archivo.

Por qué?: Ubicar los miembros bindables al principio nos permite identificar rápidamente los miembros que serán utilizados desde las vistas y facilita mucho la lectura de nuestro código.

Por qué?: Al ubicar el detalle de implemtnación de las funciones al final de nuestro archivo lleva toda la complejidad al final y nos deja al principo la parte importante que son los miembros bindables.

Por qué?: Las declaraciones de funciones son hoisted por lo tanto no hay problema alguno de invocar una función que todavía no ha sido definida.

Por qué?: No deberá preocuparse por definiciones de funciones que muevan la variable a antes o después de la variable b y puedan romper nuestro código.

Por qué?: El orden es crítico con expresiones de funciones.

Function Declarations to Hide Implementation Details

```
/**
* avoid
* Using function expressions.
*/
function Avengers(dataservice, logger) {
    var vm = this;
    vm.avengers = [];
    vm.title = 'Avengers';

    var activate = function() {
        return getAvengers().then(function() {
            logger.info('Activated Avengers View');
        });
    }

    var getAvengers = function() {
        return dataservice.getAvengers().then(function(data) {
            vm.avengers = data;
            return vm.avengers;
        });
    }

    vm.getAvengers = getAvengers;

    activate();
}
```

Function Declarations to Hide Implementation Details

```
/*
* recommend
* Using function declarations
* and bindable members up top.
*/
function Avengers(dataservice, logger) {
    var vm = this;
    vm.avengers = [];
    vm.getAvengers = getAvengers;
    vm.title = 'Avengers';

    activate();

    function activate() {
        return getAvengers().then(function() {
            logger.info('Activated Avengers View');
        });
    }

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

Defer Controller Logic

Simplifique la lógica de los controladores, delegando la misma a servicios.

Por qué?: La lógica llevada a servicios puede ser reutilizada por varios controladores que acceden a la misma a través de las funciones expuestas en los mismos.

Por qué?: La lógica aislada en servicios es más fácil de testear y su invocación desde servicios puede ser fácilmente simulado.

Por qué?: Remueve dependencias y oculta detalles de implementación en los controladores.

Defer Controller Logic

```
/* avoid */
function Order($http, $q) {
    var vm = this;
    vm.checkCredit = checkCredit;
    vm.total = 0;

    function checkCredit() {
        var orderTotal = vm.total;
        return $http.get('api/creditcheck').then(function(data) {
            var remaining = data.remaining;
            return $q.when(!!(remaining > orderTotal));
        });
    };
}
```

Defer Controller Logic

```
/* recommended */
function Order(creditService) {
    var vm = this;
    vm.checkCredit = checkCredit;
    vm.total = 0;

    function checkCredit() {
       return creditService.check();
    };
}
```

Keep Controllers Focused

Defina un controlador por vista, trate de no reutilizar el controlador en otras vistas. En su lugar, mueva lógica reutilizable dentro de servicios manteniendo los controladores simples y enfocados en las vistas.

Por qué?: La reutilización de controladores para distintas vistas puede resultar muy difícil de testear y más si la aplicación es grande.