

Hexacta Labs

AngularJS



sadosky
2009- BEST IT COMPANY
2013- LIFETIME BUSINESS
ACHIEVEMENT
2013- BEST IT SOLUTION



ExportAr
Argentina
SERVICES EXPORTER
OF THE YEAR 2011

CMMI
SM
LEVEL 3 APPRAISED

**GREAT PLACE
TO WORK**
INSTITUTE ARGENTINA
2011- 2ND PLACE
2013- 7TH PLACE

hexacta

Agenda

- > Introduccion
- > Modulos
- > Directivas
- > MVVM
- > Scope
- > Directivas custom
- > Filters
- > Services
- > IOC
- > AJAX
- > REST
- > Routing
- > Testing
- > Herramientas
- > Resumen

Introduccion

Introduccion

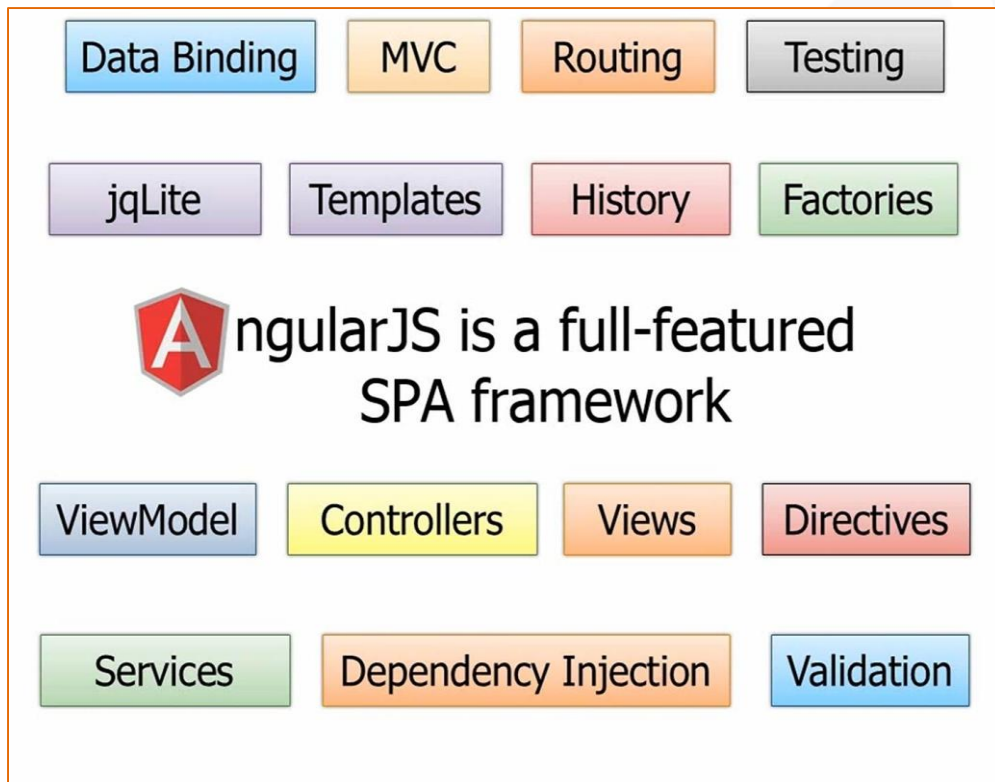
Que es AngularJS



Google

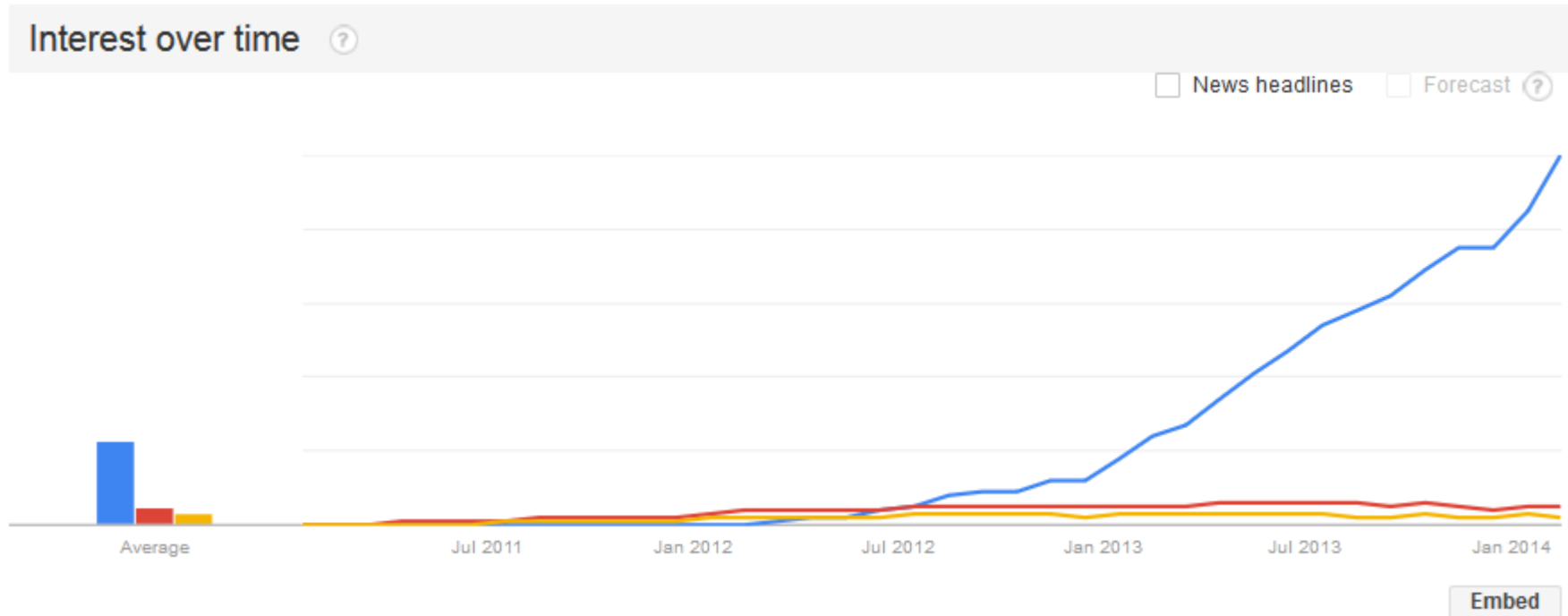
Introduccion

Que es AngularJS



Introduccion

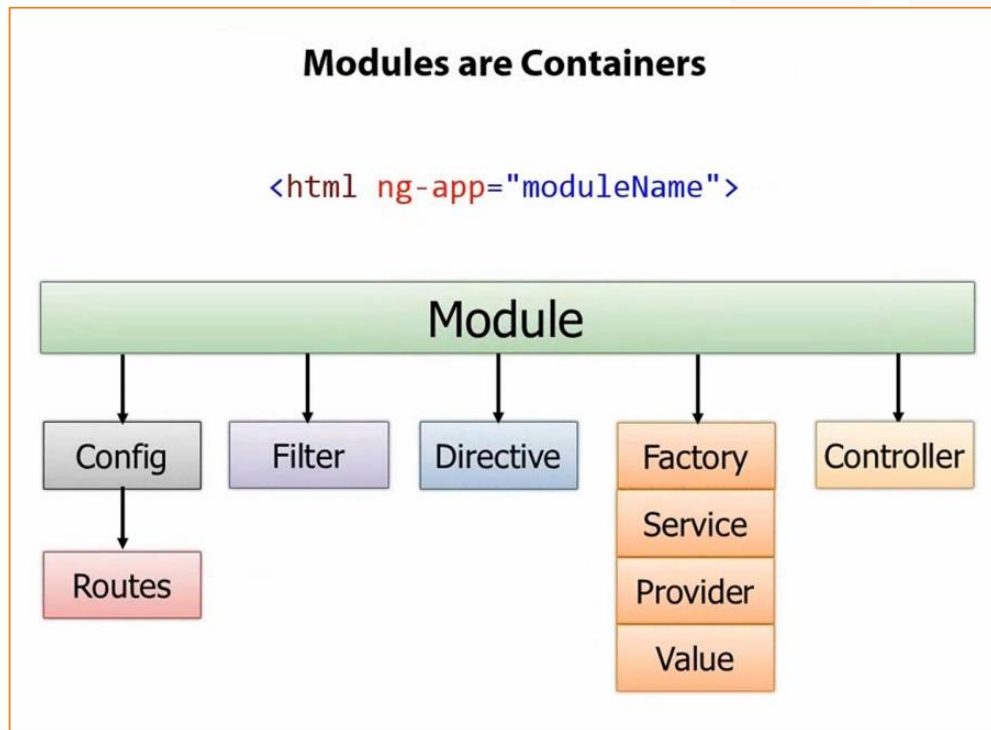
Que es AngularJS



Modulos

Modulos

Que es un modulo?



Modulos

Best practices

Se recomienda escribir:

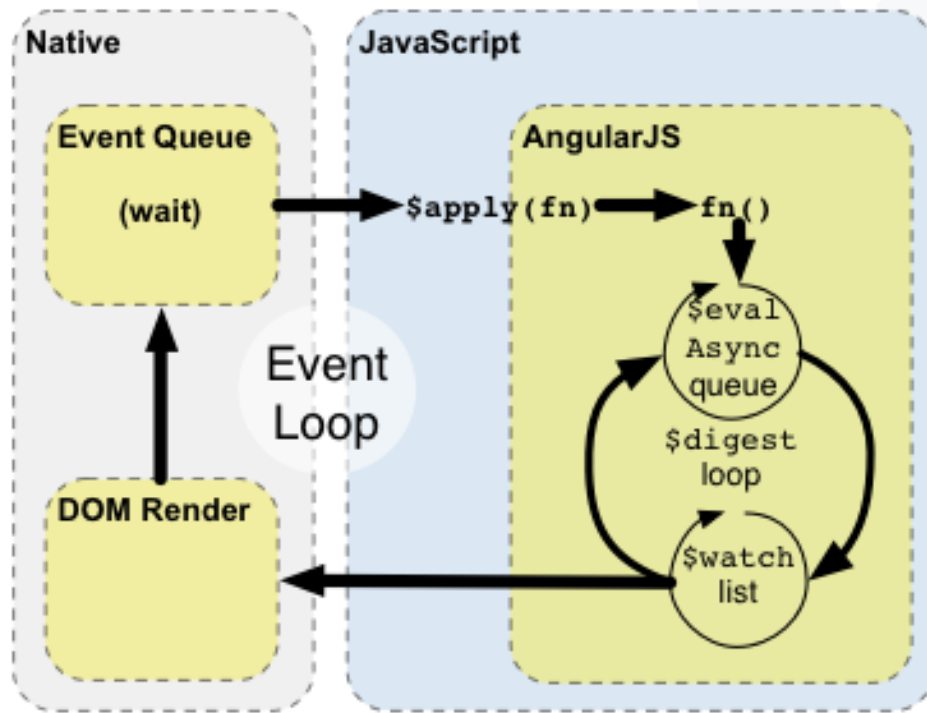
- > Un modulo por cada feature de la aplicacion
- > Un modulo por cada componente reusable
- > Un modulo a nivel de aplicacion que dependa de todos los modulos de arriba y que contenga codigo cualquier codigo de inicializacion.

Pueden encontrar una guia de Best Practices en:

<http://blog.angularjs.org/2014/02/an-angularjs-style-guide-and-best.html>

Event Loop

Event Loop



Directivas

Directivas

Que son las directivas?

- > Nos permiten agregar nuevo comportamiento a nuestro HTML!
- > Esto se realiza simplemente agregando nuevos atributos y elements

Directivas

ng-app y ng-model

Directive

```
<!DOCTYPE html>
<html ng-app="">
  <head lang="en">
    <meta charset="utf-8" />
    <title>My Angular Application!!</title>
    <script src="http://code.angularjs.org/1.2.0-rc.2/angular.js"></script>
  </head>
  <body>Write text
  <form>
    <input type="text" ng-model="text" /> {{text}}</form></body>
</html>
```

Directive

Data Binding

docs.angularjs.org/api

Aplicaciones Yahoo! LicenseServer Restart AxiomaServer Restart elonce.com Eros - Task Board - ... Outlook Web App ¡Bienvenido a Faceb... Otros marcadores

ANGULARJS Home Learn Develop Discuss

v1.2.14-build.2302+sha.f82 / API Reference

directive

- a
- form
- input
- input[checkbox]
- input[email]
- input[number]
- input[radio]
- input[text]
- input[url]
- ngApp
- ngBind
- ngBindHtml
- ngBindTemplate
- ngBlur
- ngChange
- ngChecked
- ngClass
- ngClassEven
- ngClassOdd
- ngClick
- ngCloak
- ngController
- ngCopy
- ngCsp
- ngCut
- ngDbclick
- ngDisabled
- ngFocus
- ngForm
- ngHide
- ngHref
- ngIf
- ngInclude
- ngInit
- ngKeydown
- ngKeyPress
- ngKeyUp
- ngList
- ngModel

AngularJS API Docs

Improve this doc

Welcome to the AngularJS API docs page. These pages contain the AngularJS reference materials for version **1.2.13 romantic-transclusion**.

The documentation is organized into **modules** which contain various components of an AngularJS application. These components are **directives**, **services**, **filters**, **providers**, **types**, global APIs and testing mocks.

Angular Namespaces **\$** and **\$\$** To prevent accidental name collisions with your code, Angular prefixes names of public objects with **\$** and names of private objects with **\$\$**. Please do not use the **\$** or **\$\$** prefix in your code.

Angular Namespace

ng (core module)

This module is provided by default and contains the core components of AngularJS.

This is the core collection of directives you would use in your template code to build an AngularJS application.

Directives Some examples include: `ngClick`, `ngInclude`, `ngRepeat`, etc...

Services / Factories This is the core collection of services which are used within the DI of your application. Some examples include: `$compile`, `$http`, `$location`, etc...

Filters The core filters available in the ng module are used to transform template data before it is rendered within directives and expressions. Some examples include: `filter`, `date`, `currency`, `lowercase`, `uppercase`, etc...

Global APIs The core global API functions are attached to the angular object. These core functions are useful for low level JavaScript operations within your application.

Directivas

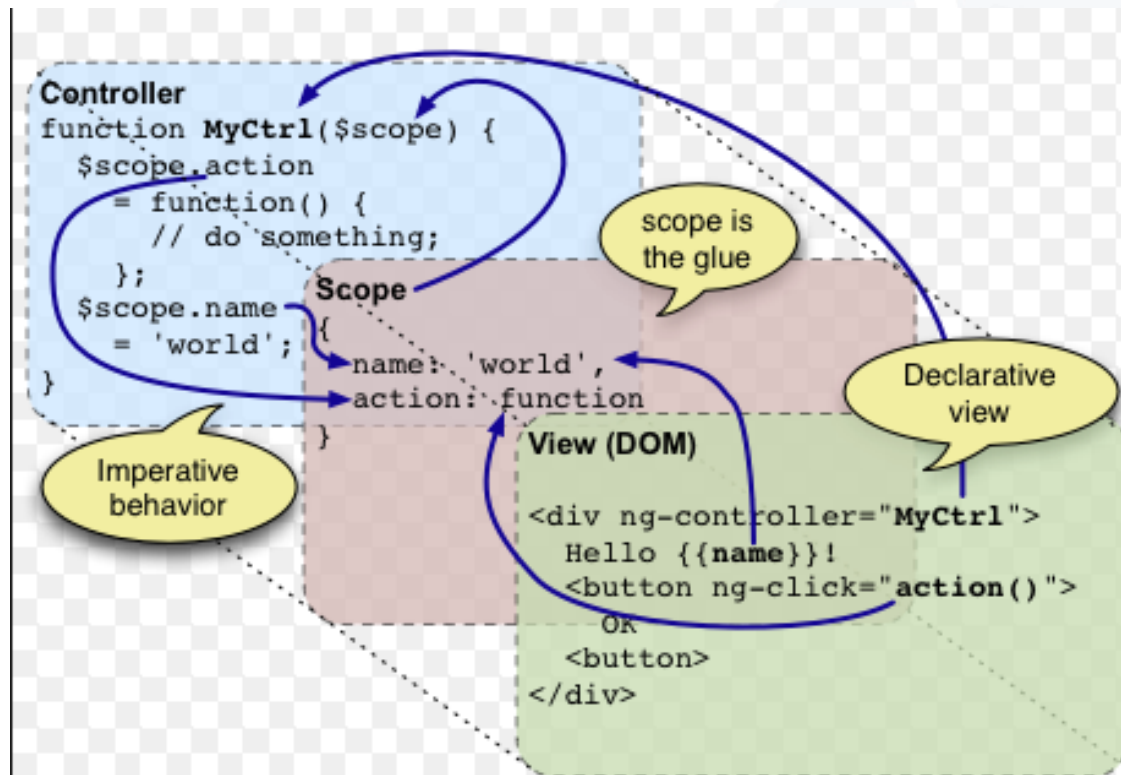
Scope



`$scope` es el “pegamento” entre el Controller y la vista

Directivas

Controller



Directivas

Controller

```
<!DOCTYPE html>
<html ng-app="myApp">
  <head lang="en">
    <meta charset="utf-8" />
    <title>My Angular Application!</title>
    <script src="http://code.angularjs.org/1.2.0-rc.2/angular.js"></script>
    <script src="./js/ejemplo2.js"></script>
  </head>
  <body>Write text
  <form>
    <div ng-controller="myCtrl">
      <input type="text" ng-model="name" /> {{name}}
    </div>
  </form>
</body>
</html>
```

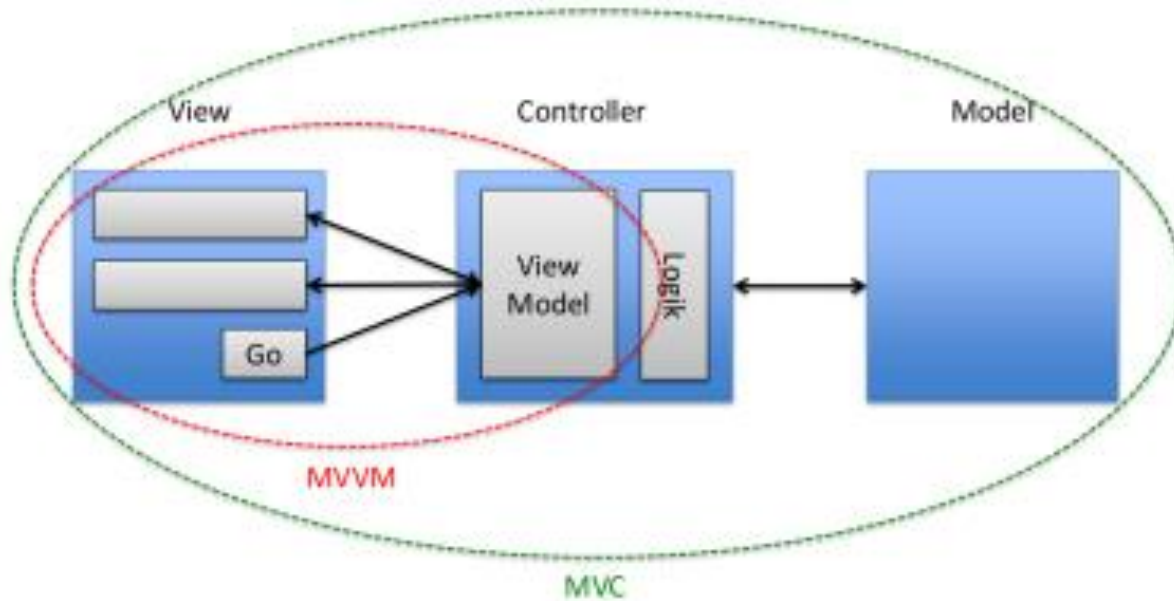
```
var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
  $scope.name = "hola"
});
```

MV*

MVVM

MVVM? MVC? MV*?



MVVM

Scope

- > Es lo que permite que la vista, el modelo y el controller trabajen de forma sincronizada.
- > Provee una manera de observar cambios en el modelo (**\$watch**) y propagar cambios a la vista (**\$apply**) (y viceversa)

Jerarquía y herencia de scopes :

- > Cada aplicación tiene un único rootScope pero puede tener varios scopes hijos que definen jerarquías.
- > El root puede recuperarse usando **\$rootScope**

```
var scope = $rootScope;
```

MVVM

Scope

> Jerarquía y herencia de scopes:

```
var padre = $rootScope;
```

```
var hijo = padre.$new();
```

```
padre.saludo = "hola";
```

```
expect(hijo.saludo).toEqual('hola');
```

```
hijo.saludo = "chau";
```

```
expect(hijo.saludo).toEqual('chau');
```

```
expect(padre.saludo).toEqual('hola');
```

Directivas Custom

Directivas Custom

Motivación

- > Nos dan la posibilidad de transformar HTML en DSL (DOMAIN SPECIFIC LANGUAGE).
Ej: `<qr-code src="qrUrl"> </qr-code>`
- > Provee una capa de abstracción la cual nos evita manipular el DOM desde los Controllers. Las directivas pueden acceder al scope del Controller en el que están definidas.
- > Un gran paso hacia el lado de la reutilización
- > Uno de los temas más heavys de Angular, acá es donde definitivamente está la pendiente máxima de la curva de aprendizaje del framework y todo el power. Qué tan difícil puede ser, ¿no?

Directivas Custom

Directive Definition Object

```
myModule.directive('directiveName', function factory(injectables) {  
  var directiveDefinitionObject = {  
    priority: 0,  
    template: '<div></div>', // or // function(tElement, tAttrs) { ... },  
    // or  
    templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },  
    replace: false,  
    transclude: false,  
    restrict: 'A',  
    scope: false,  
    controller: function($scope, $element, $attrs, $transclude, otherInjectables) { ... },  
    require: 'siblingDirectiveName', // or // ['^parentDirectiveName', '?optionalDirectiveName', '?^optionalParent'],  
    compile: function compile(tElement, tAttrs, transclude) {  
      return {  
        pre: function preLink(scope, iElement, iAttrs, controller) { ... },  
        post: function postLink(scope, iElement, iAttrs, controller) { ... }  
      }  
      // or  
      // return function postLink( ... ) { ... }  
    },  
    // or  
    // link: {  
    //   pre: function preLink(scope, iElement, iAttrs, controller) { ... },  
    //   post: function postLink(scope, iElement, iAttrs, controller) { ... }  
    // }  
    // or  
    // link: function postLink( ... ) { ... }  
  };  
  return directiveDefinitionObject;  
});
```

Directivas Custom

For Human Beings

> Vamos a explicar el ABC del tema con las propiedades que nos van a permitir poder comenzar a trabajar con ellas.

- template o templateUrl
- replace
- restrict
- scope
- link

```
myModule.directive('gravatar', function factory() {  
    var directiveDefinitionObject = {  
        template: '</img>',  
        replace: true,  
        restrict: 'E',  
        link: function postLink(scope, element) {  
            element.on('click', function clickHandler(event) {  
                alert('Mi url es: ' + scope.gravatarUrl);  
            });  
        }  
    };  
    return directiveDefinitionObject;  
});
```

Filters

```
<input type="text" ng-model="nombre" />
```

```
{{nombre | uppercase | limitTo: 3}}
```

Filters

```
angular.module('moviesApp', []).  
filter('filmadasEn', function() {  
  
    return function(peliculas, pais) {  
        var resultado = [];  
  
        peliculas.forEach(function(pelicula) {  
            if(pelicula.pais == pais){  
                resultado.push(pelicula);  
            }  
        });  
  
        return resultado;  
    };  
  
});
```

Servicios

```
module.service('MyService', function() {  
    this.method1 = function() {  
        //..  
    }  
  
    this.method2 = function() {  
        //..  
    }  
});
```

En **.service** creamos los métodos del servicio con **this.methodname**

IOC

Inyección de dependencias

- > La inyección de dependencias (DI) es un patrón de diseño de software que se ocupa de cómo el código maneja las dependencias.
- > El subsistema de inyección es el encargado de la creación de instancias de servicios, resolución de dependencias, y la provisión de dependencias a los componente que los solicitaron.
- > Existen dos clases de inyección de dependencias:
 - > Implícita
 - > Explícita

Inyección de dependencias

- > Las dependencias pueden determinarse a partir del nombre del parámetro.

```
angular.module('myServiceModuleDI', []).  
  factory('notify', function($window) {  
    var msgs = [];  
    return function(msg) {  
      msgs.push(msg);  
      if (msgs.length == 3) {  
        $window.alert(msgs.join("\n"));  
        msgs = [];  
      }  
    };  
  }).  
  controller('MyController', function($scope, notify) {  
    $scope.callNotify = function(msg) {  
      notify(msg);  
    };  
  });
```

Inyeccion de dependencias

Un componente debe explícitamente definir sus dependencias usando uno de los métodos de inyección:

Inline array injection annotation:

```
myModule.controller('MyController', ['$location',  
function($location) {...}]);
```

\$inject property:

```
var MyController = function($location) { ... };  
MyController.$inject = ['$location'];  
myModule.controller('MyController', MyController);
```

AJAX

AJAX

Promise

- > Proporciona una interfaz bien definida para interactuar con un objeto que representa el resultado de una acción que se realiza de forma asíncrona.
- > Mediante la utilización de una interfaz estándar, diferentes componentes pueden devolver promise de acciones asíncronas y los consumidores pueden utilizar los promise de una manera predecible.

\$http retorna:

- > success : **function(data, status, headers, config)**
- > error: **function(data, status, headers, config)**
- > then: **function(data, status, headers, config)** (siempre)

AJAX

Promise

- > \$q es la implementación de promise/deferred
- > \$q posee una API para Deferred con la cual podre crear un objeto deferred (\$q.defer()) para exponer una instancia de promise
- > \$q posee una API de promise (deferred.promise) con los metodos THEN, CATH, FINALLY. Esto es porque \$q posee una gestion de errores.
- > El porque de la explicación de \$q en AJAX es porque administra llamadas asincronicas

> Ejemplo:

```
var
    defer1 = $q.defer(),
    defer2 = $q.defer();

$q.all([defer1.promise, defer2.promise])
    .then(function (results) {
        //array results
    });

defer1.resolve();
defer2.resolve();
```

AJAX

Promise

- > De los metodos que expone utilizaremos `.all()` en el caso de que necesitemos conocer cuando terminen varias llamadas `$http`.
- > Aca tenemos un ejemplo de `$http` y `$q`:

```
var
    request1 = $http.get('/someUrl1'),
    request2 = $http.get('/someUrl2');

var mypromise = $q.all([request1, request2]);

mypromise.then(function (results) {
    //array results
});
```

REST

REST

Qué es?

- > AngularJS provee el servicio `$resource` para interactuar con servicios REST
- > Usa el servicio `$http` pero expone una interface a mas alto nivel
- > Provee los siguientes metodos por default:
 - `get`: GET
 - `save`: POST
 - `delete`: DELETE
 - `query`: GET
- > La diferencia entre `get` y `query` es que `get` espera un unico objeto de JSON `{}`. En cambio `query` espera una colleccion `[]`.
- > Podemos agregar, redefinir y configurar cada metodo.

Ejemplo

```
var movies = $resource('http://server/movies');  
var movie = {title: 'Scarface', year: 1982};  
movies.save(movie);  
movies.delete(movie);  
var movies2011 = movies.query({year: 2011});
```

Routing

Routing

/peliculas
PeliculaController

Búsqueda

Título:

Buscar

Título	Año	Rating
Descripción:		

index.html

MainController

Portal de Películas

Home Películas Series

CONTENIDO

Portal de Películas | Hexacta 2014

/detalle
DetalleController

Título



★★★★★

Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido. Descripción larga sobre la película y su contenido.

Routing

```
var app = angular.module('PelículasApp', ['ngRoute']);
app.config(['$routeProvider', function($routeProvider) {
  $routeProvider.
    when('/películas', {
      templateUrl: 'views/películas.html',
      controller: 'PelículaController'
    }).
    when('/detalle', {
      templateUrl: 'views/detalle.html',
      controller: 'DetalleController'
    }).
    otherwise({
      redirectTo: '/películas'
    });
}]);
```

Routing

```
<a href="peliculas">Películas</a>
```



```
when('/películas', {  
  templateUrl: 'views/películas.html',  
  controller: 'PelículaController'  
})
```

Routing

Index.html

```
<div><!-- Menu --></div>  
<div ng-view></div>  
<div><!-- Footer --></div>
```




```
when('/peliculas', {  
  templateUrl: 'views/peliculas.html',  
  controller: 'PeliculaController'  
})
```

Routing

peliculaController.js

```
app.controller('PeliculaController', function ($scope) {  
    $scope.peliculas = [];  
    $scope.titulo = '';  
});
```

```
when('/peliculas', {  
    templateUrl: 'views/peliculas.html',  
    controller: 'PeliculaController'  
})
```



```
when('/dette/:nombre', {  
    templateUrl: 'views/dette.html',  
    controller: 'DetteController'  
})
```


Routing

```
app.controller('DetalleController',  
    function ($scope, $routeParams) {  
  
        $scope.nombreDeLaPelicula = $routeParams.nombre;  
        ...  
  
    }) ;
```

Routing

```
app.controller('DetalleController',  
    function ($scope, $location) {  
        // '/detalle'  
        var path = $location.path();  
        $scope.back = function() {  
            $location.path('/peliculas');  
        }  
    }  
);
```

Testing

Testing

Unit testing

- > Para hacer unit testing con AngularJS podemos usar Jasmine.
- > Jasmine es una libreria para hacer testing basado en BDD.
- > Los tests se implementan con un lenguaje simple y facil de entender.
- > <https://github.com/angular/angular.js/blob/master/test/ng/directive/ngClassSpec.js>

Testing

Integration testing

- > Para hacer integration testing (o E2E testing) con AngularJS podemos usar Protractor.
- > Protractor esta basado en Jasmine para la implementacion de los tests. Esto permite usar la misma sintaxis para todos los tests.
- > Protractor usa Webdriver para emular el input del usuario en la aplicacion web.
- > <https://github.com/angular/angular.js/blob/master/test/e2e/docsAppE2E.js>

Herramientas

NodeJS

NodeJS



- > NodeJS es una plataforma basada en el runtime Javascript de Chrome.
- > Permite implementar aplicaciones en Javascript que manejen tareas como IO, eventos, requests y responses, y cualquier otra tarea típicamente asociada a un servidor o aplicación de backend

NPM



- > Node Package Manager es un package manager :O
- > Se encargar de resolver dependencias a la hora de instalar un modulo de node.
- > Todas las herramientas que vemos a continuación se instalan a través de NPM.
- > Es cross plataforma

Herramientas

Grunt



- > Grunt es una herramienta para automatizar tareas.
- > Por ejemplo: Levantar la aplicación, correr tests, empaquetar, minificar, procesar imágenes, css, etc.
- > Las tareas se pueden configurar y definir en el archive Gruntfile.js

KarmaJS



- > KarmaJS se encarga de automatizar la corrida de los tests.
- > Administra donde correr los tests.
- > Delega la escritura de los casos de test en otros frameworks como Jasmine.

Herramientas

JasmineJS



- > Permite implementar casos de prueba en un lenguaje mas natural.
- > Orientado a BDD.

JSHint



- > JSHint es una herramienta de analisis de codigo
- > Es configurable y permite detectar problemas en el codigo.

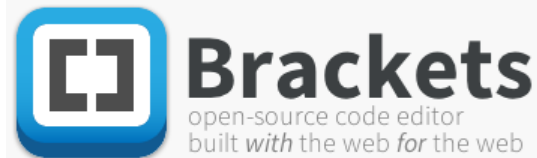
Herramientas

Yeoman



- > Herramienta de arquetipos templates generators.
- > Simplifica la creación de una aplicación en base a distintos generators customizables, permitiendo usar todas las herramientas mencionadas con una configuración típica.

Brackets



- > Brackets es un IDE para Javascript, hecho con ❤️ y Javascript.
- > Tiene facilidades para Javascript, HTML, CSS.
- > Integra varias de las herramientas nombradas anteriormente para facilitar el desarrollo

Protractor



- > Permite implementar tests de integracion
- > Usa Jasmine como lenguaje y WebDriver para interactuar con el browser

Bower



- > Bower es un package manager.
- > A diferencia de NPM, Bower se encarga de las dependencias que va a usar nuestra aplicación web cuando se ejecute (jQuery, Bootstrap, AngularJS, etc).

Resumen

Resumen

Pros vs. Contras

- > Calidad en terminos de estructura y robustez
- > HTML declarativo. Un desarrollador puede entender el comportamiento de una pagina sin leer 4000 lineas de Javascript
- > Componentes reusables
- > Patrones MVC/MVVM
- > Promueve el uso de TDD, End2End integration testing / unit testing
- > Amplia el HTML con el uso de directivas
- > Pone enfasis en separacion de responsabilidades. Por ejemplo: llamadas AJAX y manipulacion del DOM no deberian estar acoplados

Resumen

Pros vs. Contras

- > Provee inyección de dependencias simplificando el código y testing
- > Usa una versión lite de jQuery para soporte cross browser
- > Tiene un sistema de templates para reusar código de la vista
- > Two Way Data-Binding. Una aplicación web típica puede contener hasta 80% de código dedicado a recorrer, manipular y escuchar eventos del DOM. Data-Binding elimina este código y el desarrollador se concentra en la lógica y no en cómo presentar la información al usuario.

Preguntas?

ARGENTINA

Clay 2954

Buenos Aires (C1426DLD)

tel: 54+11+5299 5400

BRASIL

Cardoso de Melo 1470 – 8, Vila Olimpia

San Pablo (04548004)

tel: 55+11+3045 2193

URUGUAY

Roque Graseras 857

Montevideo (11300)

tel: 598+2+7117879

USA

12105 Sundance Ct.

Reston (20194)

tel:+703 842 9455



HexactaArg



@Hexacta



www.hexacta.com