



COLEGIO OFICIAL DE
INGENIEROS EN INFORMÁTICA
DE LA COMUNIDAD VALENCIANA



Desarrollo de aplicaciones web con Angular JS

Objetivos

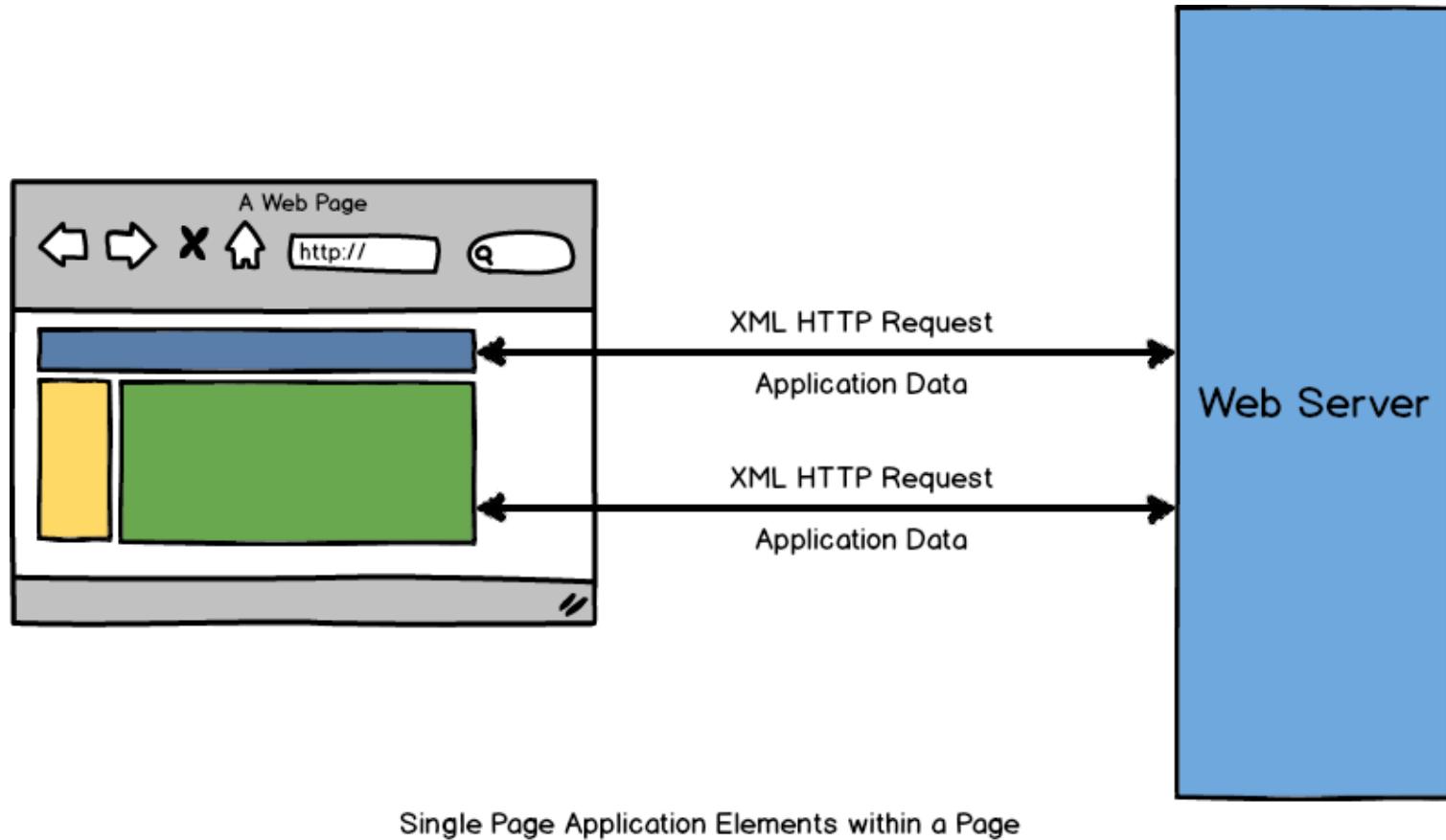
- Conocer la estructura básica de una aplicación en angular
- Conocer y comprender la funcionalidad de un controlador en angular
- Conocer y comprender la funcionalidad de un servicio en angular
- Conocer y comprender la funcionalidad de un filtro en angular
- Conocer y comprender la funcionalidad de una directiva en angular
- Conocer y comprender lo que significa una aplicación web basada en estados

Conceptos a desarrollar

- **MVC:** patrón de arquitectura que separa en capas diferentes la lógica de datos, la lógica de negocio y la interfaz de usuario
- **SPA:** patrón de arquitectura que implementa el MVC en cliente
- **Angularjs:** framework de desarrollo SPI basado en el lenguaje javaScript
- **Módulo:** conjunto de contenedores angular
- **Contenedores angular:** funciones que pueden ser de los siguientes tipos
 - **Controlador:** funciones encargadas de recoger los eventos de la interfaz de usuario y de plasmar las consecuencias de estos
 - **Servicio:** función singleton que puede contener un valor javaScript, un objeto o una clase
 - **Filtro:** función que transforma un objeto de entrada en otro, dependiendo de unas reglas definidas
 - **Directiva:** conjunto de controlador mas vista agrupados en una etiqueta/ atributo html
- **Inyección de dependencias.**

Aplicaciones SPA

- Single Page Application



Aplicaciones hechas con Angular

<https://builtwith.angularjs.org/>

[https://www.eduonix.com/blog/
web-programming-tutorials/
top-15-websites-and-apps-built-with-angularjs/](https://www.eduonix.com/blog/web-programming-tutorials/top-15-websites-and-apps-built-with-angularjs/)

<https://w3techs.com/technologies>

Versión de Angular

```
> angular
< - ▼ Object ⓘ
  ► $$csp: function ()
  ► $$minErr: function v(b)
  ► $interpolateMinErr: function ()
  ► bind: function uc(b,a)
  ► bootstrap: function (node, modules)
  ► callbacks: Object
  ► copy: function ua(b,a,c,d)
  ► element: function (a,b)
  ► equals: function ka(b,a)
  ► extend: function S(b)
  ► forEach: function n(b,a,c)
  ► fromJson: function vc(b)
  ► getTestability: function Zd(b)
  ► hint: EventEmitter
  ► identity: function Va(b)
  ► injector: function ch(b,a)

  ► angular.version
  < - ▼ Object ⓘ
    codeName: "jaracimrman-existence"
    dot: 0
    full: "1.4.0"
    major: 1
    minor: 4
    ► __proto__: Object
```

⋮ Console

Identificar librerías

<https://wappalyzer.com/>

 **Wappalyzer**
ofrecido por <https://wappalyzer.com>

★★★★★ (899) | [Herramientas para desarrolladores](#) | 419.931 usuarios

+ AÑADIR A CHROME 

DESCRIPCIÓN GENERAL	OPINIONES	AYUDA	RELACIONADOS	
---------------------	-----------	-------	--------------	--



Compatible con tu dispositivo

Identifies software on the web

Wappalyzer is a browser extension that uncovers the technologies used on websites. It detects content management systems, web shops, web servers, JavaScript frameworks, analytics tools and many more.

List of applications Wappalyzer detects:
<http://wappalyzer.com/applications>

Firefox version:
<https://addons.mozilla.org/en-US/firefox/addon/wappalyzer/>

[Run report and connections](#)

 Sitio web
 Notificar uso inadecuado

Información adicional

Versión: 2.49
Última actualización: 3 de septiembre de 2016

¿Qué es AngularJS?

- Es un framework de desarrollo web en JavaScript.
- Mantenido por Google.
- Multiplataforma.
- Open source.
- Típicamente empleado para hacer SPA: SinglePage Apps.
- <http://angularjs.org>

Principios

- No manipular el DOM directamente como se hace con jQuery
- Todo bien separado
- Pensado para ser testeado
- Escribir menos código

Principios

- Separación "Model View Whatever"
- Data binding
- Inyección de dependencias
- Plain javascript
- Preparado para REST
- Componentes reusables

Descarga AngularJS



HTML enhanced for web apps!



Download AngularJS 1



(1.5.8 / 1.2.30)

Try the new Angular 2



View on GitHub



Design Docs & Notes

¿Qué es AngularJS?

Download AngularJS

X

Branch

1.5.x (stable)

1.2.x (legacy)



Build

Minified

Uncompressed

Zip



CDN

<https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js>



Bower

bower install angular#1.5.8



npm

npm install angular@1.5.8

Extras

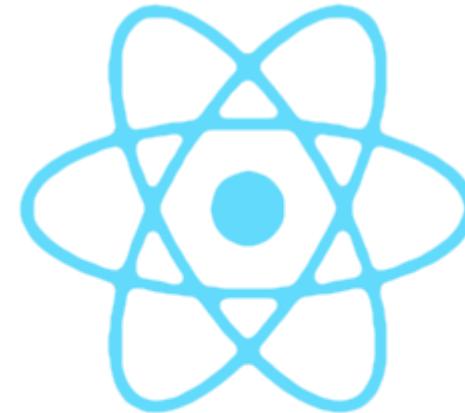
[Browse additional modules](#)

[Previous Versions](#)



[Download](#)

Otros frameworks



React



BACKBONE.JS

Herramientas

- AngularJS
- NodeJS
- Editor brackets
- Chrome Browser
- Batarang plugin

NodeJS



[HOME](#) | [ABOUT](#) | [DOWNLOADS](#) | [DOCS](#) | [FOUNDATION](#) | [GET INVOLVED](#) | [SECURITY](#) | [NEWS](#)

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries in the world.

[Download for Windows \(x64\)](#)

[v6.9.1 LTS](#)

Recommended For Most Users

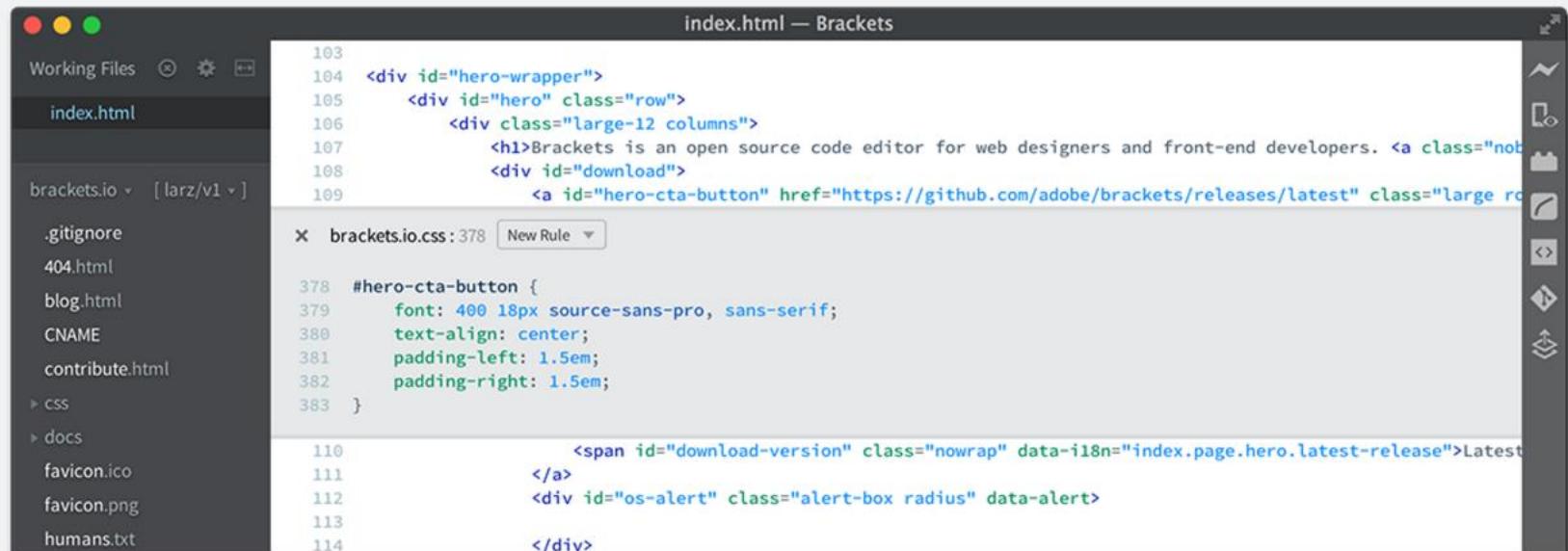
[v7.0.0 Current](#)

Latest Features

Brackets

[Download Brackets 1.7](#)

[Other Downloads](#)



The screenshot shows the Brackets IDE interface. The title bar says "index.html — Brackets". The left sidebar lists "Working Files" including "index.html", "brackets.io [larz/v1]", ".gitignore", "404.html", "blog.html", "CNAME", "contribute.html", "CSS", "docs", "favicon.ico", "favicon.png", and "humans.txt". The main editor area displays the content of "index.html". A CSS panel on the right shows the "brackets.io.css" file with the following code:

```
103 <div id="hero-wrapper">
104   <div id="hero" class="row">
105     <div class="large-12 columns">
106       <h1>Brackets is an open source code editor for web designers and front-end developers. <a class="nobr" href="https://github.com/adobe/brackets/releases/latest" id="download">
107         <div id="hero-cta-button" href="https://github.com/adobe/brackets/releases/latest" class="large radius">
108           Download Now
109         </div>
110       </div>
111     </div>
112   </div>
113 </div>
114 <span id="download-version" class="nowrap" data-i18n="index.page.hero.latest-release">Latest Version
115 </span>
116 </div>
117 <div id="os-alert" class="alert-box radius" data-alert>
118   Brackets is now available for Mac OS X!
119 </div>
```

Batarang

AngularJS Batarang
ofrecido por [AngularJS](#)

★★★★★ (1351) | [Herramientas para desarrolladores](#) | 319.585 usuarios

AÑADIDO A CHROME

DESCRIPCIÓN GENERAL OPINIONES RELACIONADOS 1.101

AngularJS Batarang

AngularJS Batarang is a Chrome extension that extends the Developer Tools, adding tools for debugging and profiling AngularJS applications. It provides a live preview of your application's state, a timeline for performance analysis, and a comprehensive set of developer tools for inspecting and modifying your code.

The screenshot shows the extension in action, displaying a live preview of an AngularJS application's state, a timeline showing the execution of code, and a detailed view of the application's codebase.

Compatible con tu dispositivo

Extends the Developer Tools, adding tools for debugging and profiling AngularJS applications.

Sitio web

Notificar uso inadecuado

Información adicional

Versión: 0.10.7

Última actualización: 7 de marzo de 2016

Tamaño: 405KIB

Idioma: English

Batarang

The screenshot shows a web browser window displaying the "Google Phone Gallery" application at localhost:8000/app/index.html#/phones. The page lists two devices:

- Motorola XOOM™ with Wi-Fi**:
The Next, Next Generation Experience the future with Motorola XOOM with Wi-Fi, the world's first tablet powered by Android 3.0 (Honeycomb).
- MOTOROLA XOOM™**:
The Next, Next Generation Experience the future with MOTOROLA XOOM, the world's first tablet powered by Android 3.0 (Honeycomb).

The Batarang Chrome extension is open, providing a debugger interface for the AngularJS application. The "Models" tab is selected, showing the following information:

- Scopes**: A tree view of the scope hierarchy:

```
< Scope (002)
  < Scope (003)
    < Scope (005) // Selected
    < Scope (007)
    < Scope (009)
    < Scope (008)
    < Scope (00D)
    < Scope (00F)
    < Scope (00H)
    < Scope (00J)
    < Scope (00L)
    < Scope (00N)
    < Scope (00P)
    < Scope (00R)
```
- Models for (005)**: A detailed view of the selected scope, showing the model definition for the first tablet:

```
{
  phone: {
    age: 0
    id: motorola-xoom-with-wi-fi
    imageUrl: img/phones/motorola-xoom-with-wi-fi.0.jpg
    name: Motorola XOOM™ with Wi-Fi
    snippet: The Next, Next Generation

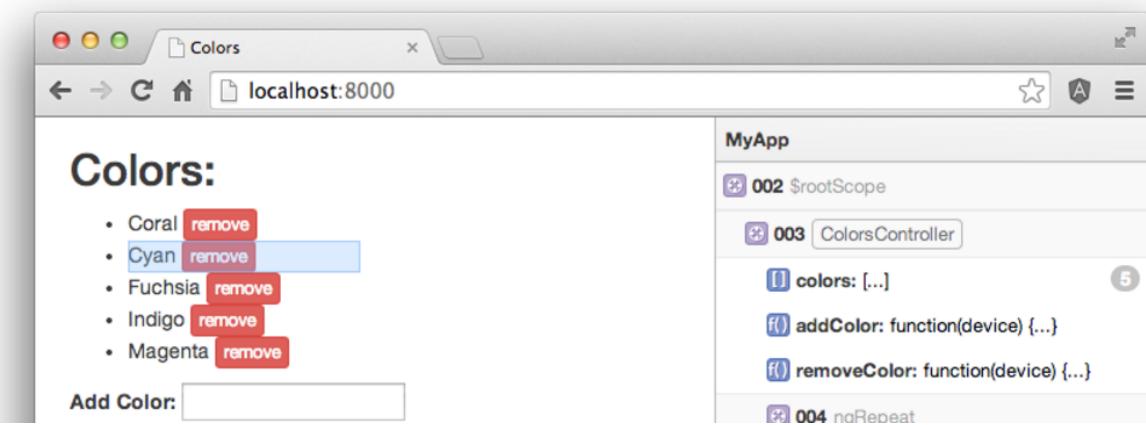
    Experience the future with Motorola XOOM with Wi-Fi, the
    world's first tablet powered by Android 3.0 (Honeycomb).
  }
}
```

Ng inspector



ng-inspector

The AngularJS inspector pane for your browser



Servidor web

- Node
- Instalar
 - **npm install http-server -g**
- Ejecutar
 - **http-server -p <port>**



ANGULAR

10148 results for 'ANGULAR'

angular angularcore

HTML enhanced for web apps

★ 182 v1.5.8

↳ angular, framework, browser, client-side

angular-selectable jonotron

An angular module for selecting things

★ 0 v0.0.8

↳ angular

angular-zelda emarsys-deployer

Angular Zelda

★ 0 v2.1.1

↳ angular, zelda



Bower



Bower is a command line utility. Install it with npm.

```
$ npm install -g bower
```

Bower requires [node](#), [npm](#) and [git](#).

Latest release: [npm v1.7.9](#)

For troubleshooting installation on different platforms, read the [troubleshooting](#) wiki page.

Getting started

Install packages

Install packages with [bower install](#). Bower installs packages to [bower_components/](#).

```
$ bower install <package>
```

Bower

angular-latest

HTML enhanced for web apps

angular

53011

angular-ui-bootstrap

Native AngularJS (Angular) directives for Bootstrap. Smaller footprint (20kB gzipped), no 3rd party JS dependencies (jQuery, bootstrap JS) required. Please read the README.md file before submitting an issue!

angular-loading-bar

chieffancypants 4855

A fully automatic loading / progress bar for your angular apps.

angular-toastr

Foxandxss 1044

Angular port of CodeSeven/toastr.

angular

Bower

- Generar bower.json en nuestro proyecto
- **bower init**

```
? name HolaMundo
? description Website to test the libraries of architecture thin
? main file
? what types of modules does this package expose?
? keywords
? authors jmortega <jmoc25@gmail.com>
? license MIT
? homepage
? set currently installed components as dependencies? Yes
? add commonly ignored files to ignore list? Yes
? would you like to mark this package as private which prevents it fr

{
  name: 'HolaMundo',
  description: 'Website to test the libraries of architecture thin',
  main: '',
  authors: [
    'jmortega <jmoc25@gmail.com>'
  ],
  license: 'MIT',
  moduleType: [],
  homepage: '',
  ignore: [
    '**/*',
    'node_modules'
```

Bower

- Instalar angular como dependencia
- **bower install angular --save**

```
bower angular#*           cached https://github.com/angular/bower-angular.git#1.5.6
bower angular#*           validate 1.5.6 against https://github.com/angular/bower-angular.git#1.5.6
bower angular#*             new version for https://github.com/angular/bower-angular.git#1.5.6
bower angular#*             resolve https://github.com/angular/bower-angular.git#1.5.6
bower angular#*             checkout v1.5.8
bower angular#*             resolved https://github.com/angular/bower-angular.git#1.5.8
bower angular#*             install angular#1.5.8

angular#1.5.8 bower_components\angular
```

Módulos

<http://ngmodules.org/>

ANGULAR MODULES
FIND MODULES FOR ANGULARJS

Search

[Sign in with Github](#)

Modules [Submit a Module](#) [Tags](#) [Blog](#) [Give Feedback](#)

[Popular](#) [Newest](#) [A-Z](#)

2022 modules

ng-file-upload — [danialfarid](#) 693 people use it
Lightweight Angular directive to upload files with optional FileAPI shim for cross browser support
[file drag&drop](#) [data-url](#) [image paste](#) [directive](#) [upload](#) [file-upload](#) [HTML5](#)
[FileAPI](#)

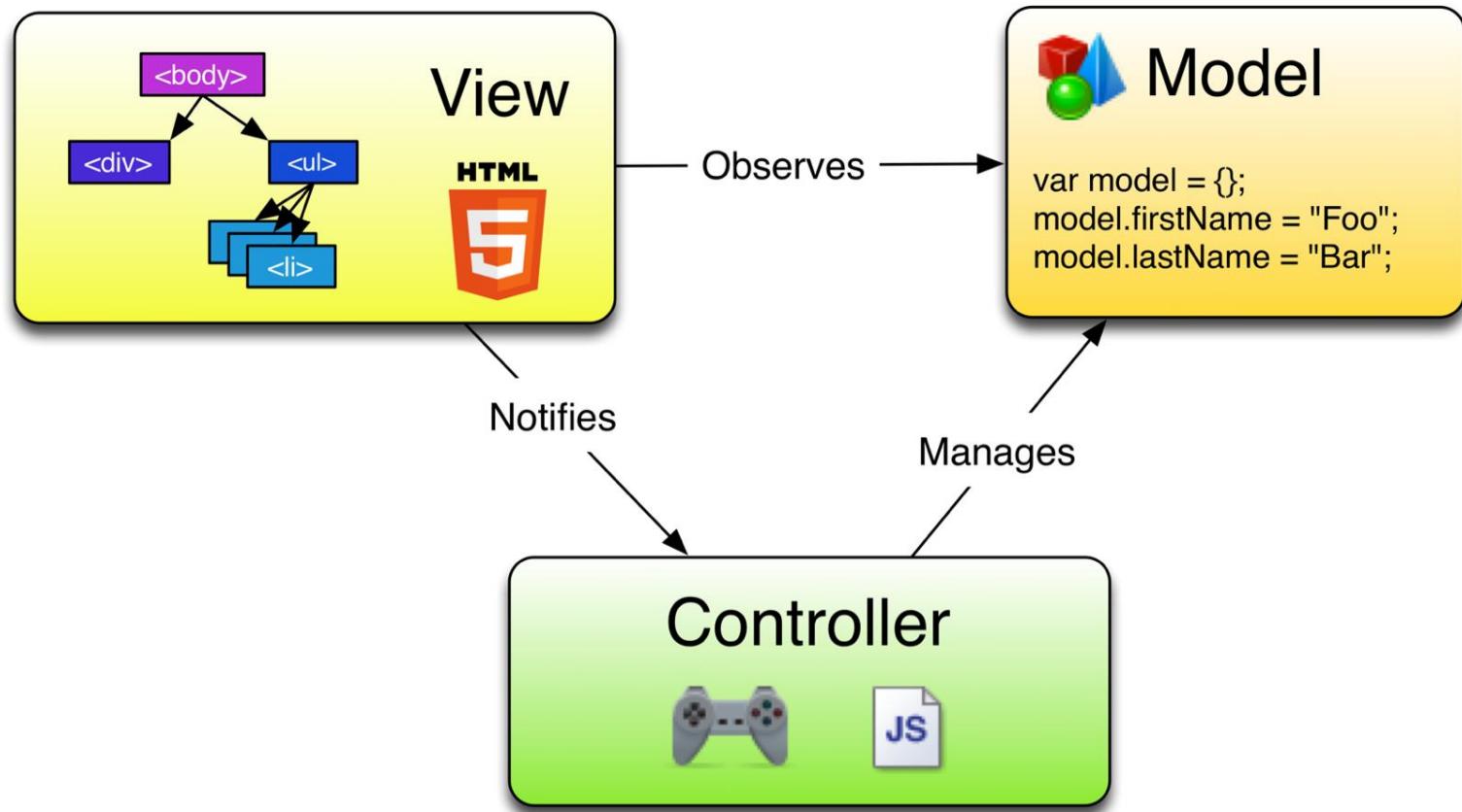
angular-ui — [angular-ui](#) 472 people use it
AngularUI - The companion suite for AngularJS
[bootstrap](#) [jQuery](#) [maps](#) [directive](#) [service](#) [ui](#)

bootstrap — [angular-ui](#) 328 people use it
Directives specific to twitter bootstrap
[bootstrap](#) [ui](#) [directives](#)

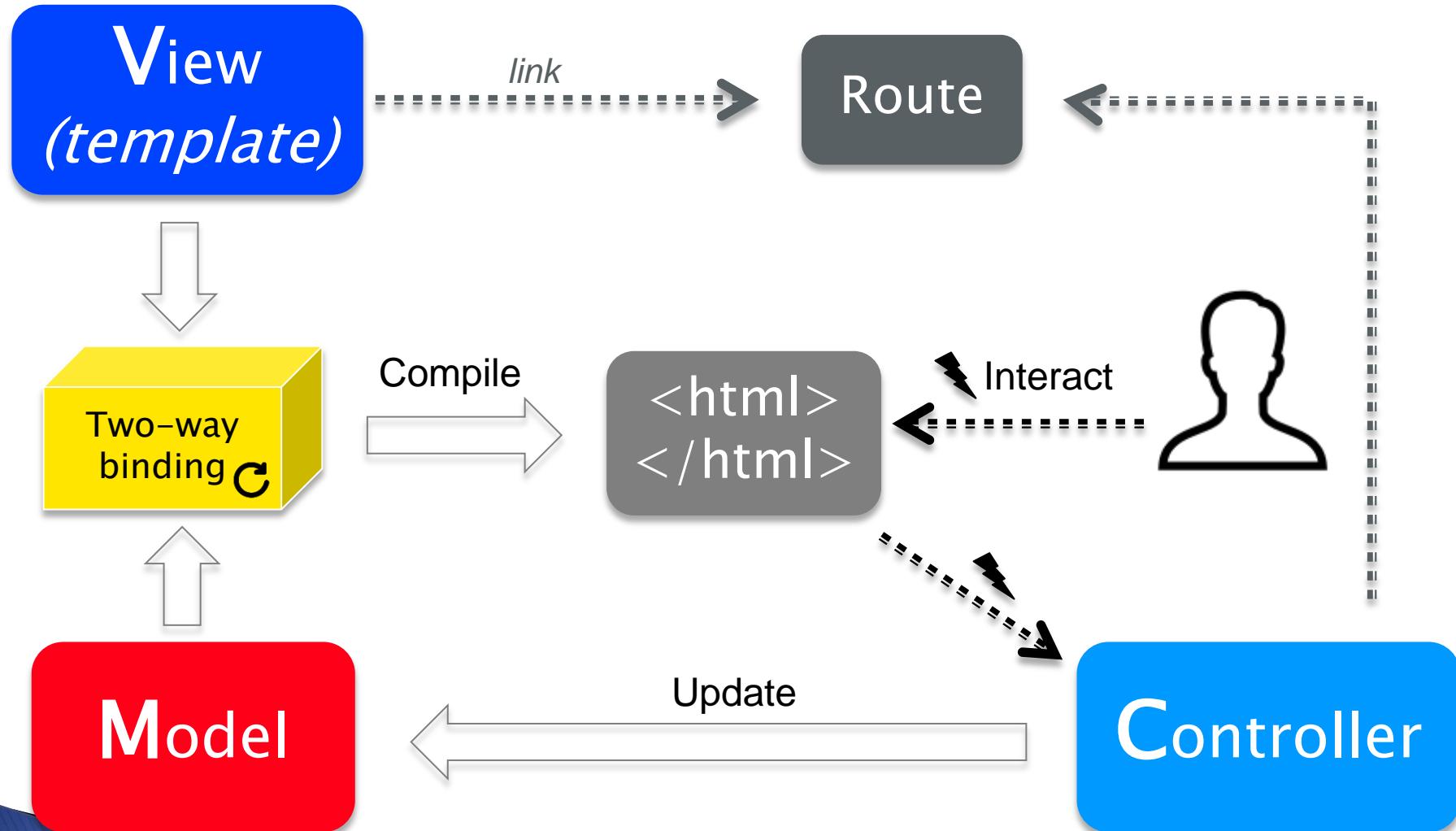
[Add a Module](#)

Popular Tags

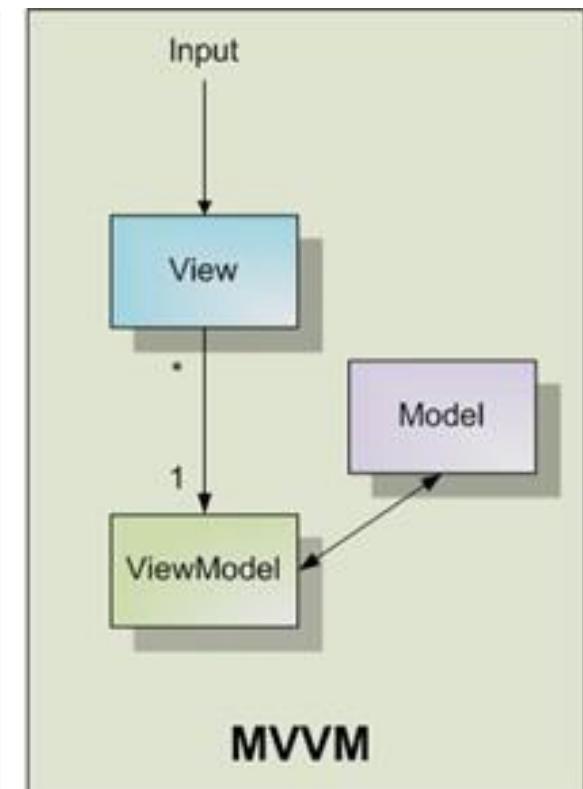
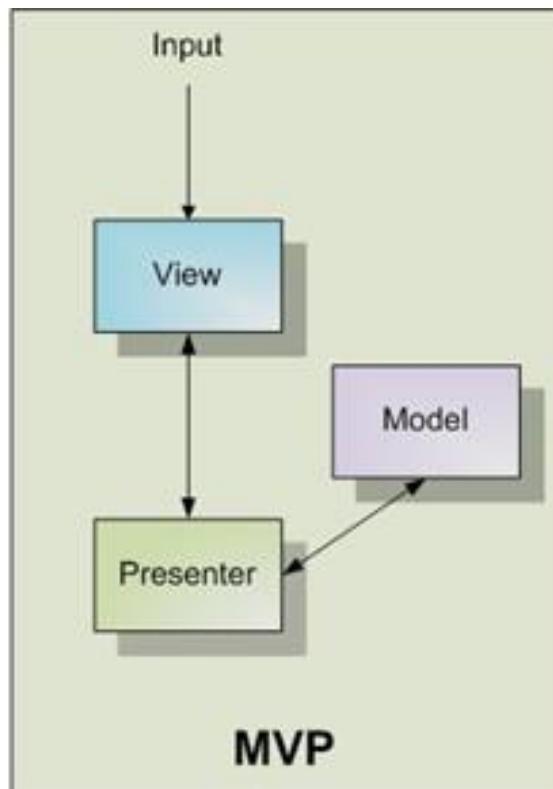
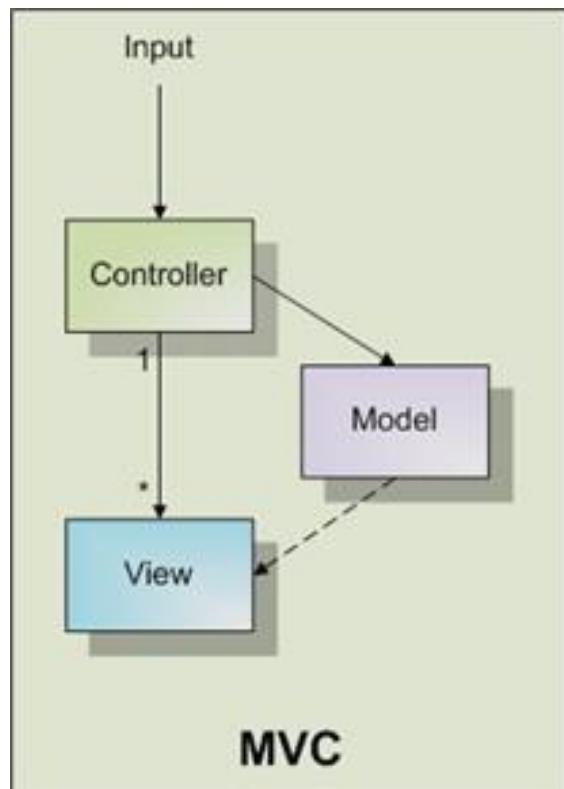
- [directive \(862\)](#)
- [angular \(232\)](#) [service \(160\)](#)
- [AngularJS \(109\)](#)
- [bootstrap \(97\)](#) [api \(92\)](#)
- [ui \(84\)](#) [input \(65\)](#)
- [module \(60\)](#) [form \(60\)](#)
- [filter \(56\)](#) [validation \(45\)](#)
- [REST \(42\)](#) [image \(42\)](#)
- [jQuery \(40\)](#) [table \(40\)](#)
- [date \(38\)](#) [select \(36\)](#)
- [scroll \(36\)](#) [angular2 \(36\)](#)
- [http \(35\)](#) [mobile \(33\)](#)



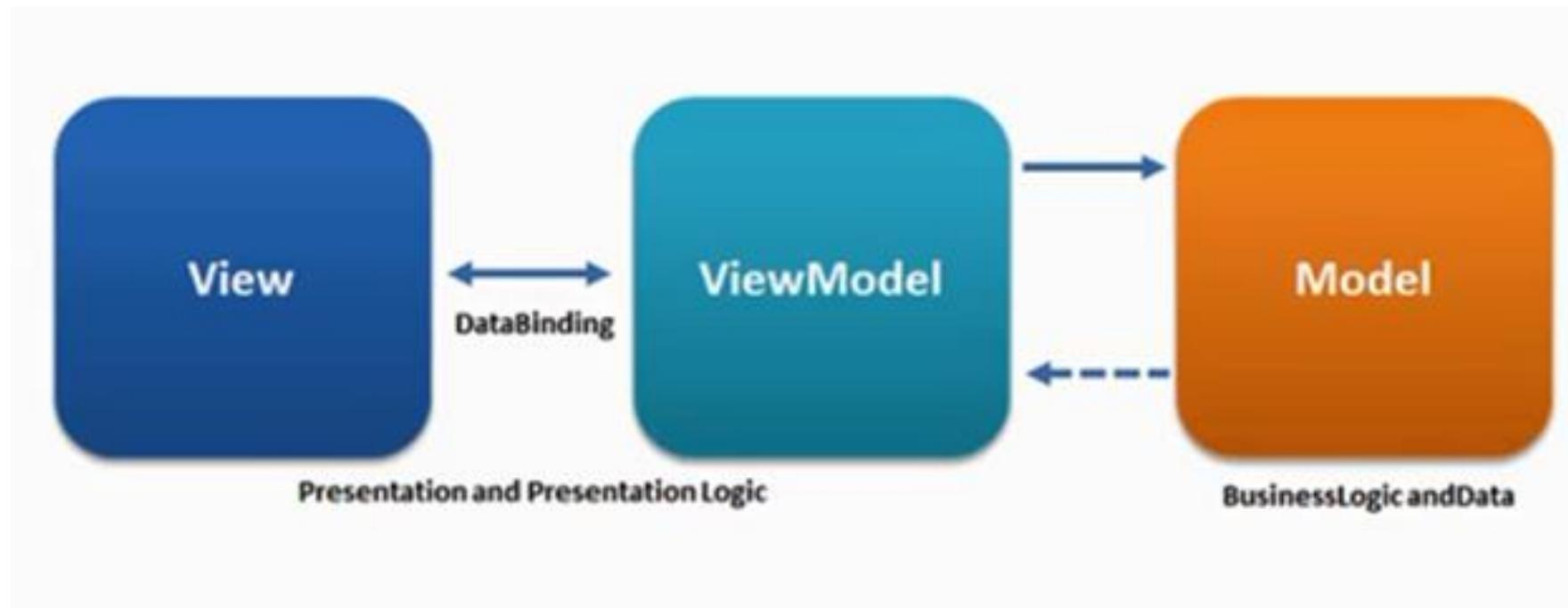
Arquitectura MVC



MVC, MVP y MVVM



MVC, MVP y MVVM



MVC, MVP y MVVM

- **Model(Scope)**
 - Información del sistema con la que se hace binding en la parte visual
- **View**
 - Define el aspecto visual o interfaz de usuario
- **ViewModel**
 - Expone los datos del modelo a la vista
 - Lógica de presentación de la ventana para gestionar el flujo
 - Comunicación con servicios que se inyectan

MVC, MVP y MVVM

view

```
<!DOCTYPE html>
<html>

  <head>
    <script data-require="angular.js@*" data-semver="1.3.0-beta.5" src="http://code.angularjs.org/1.3.0-beta.5/angular.js">
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>

  <body ng-app>
    <div ng-controller="HelloCtrl">
      Say hello to: <input type="text" ng-model="name"><br>
      <h1>Hello, {{name}}!</h1>
    </div>
  </body>
</html>
```

Binding to model

Controller

Model

```
var HelloCtrl = function ($scope) {
  $scope.name = 'World';
}
```

Hola mundo

```
<html ng-app>
<head>
<script src="js/angular.min.js"></script>
</head>
<body>
<div>
<label>Nombre:</label>
<input type="text" ng-model="nombre" ng-init="nombre =
'mundo'">
<hr>
<h1>Hola, {{nombre}}</h1>
<h1>Hola, <span ng-bind="nombre"></span></h1>
</div>
</body>
</html>
```

Directiva

Directiva

Expression

Directivas básicas

- **ng-app**: indica qué elemento es el propietario de la app en AngularJS (podría ser un div dentro de una web generada de otra forma).
- **ng-model**: enlaza el elemento con su modelo para conseguir el "double-binding" (de la vista al modelo y del modelo a la vista, automáticamente).
- **ng-init**: inicializa valores de AngularJS. Normalmente usaremos un controlador, pero a veces viene bien.
- **ng-bind**: sustituye la propiedad innerHTML del elemento por el valor de la variable en AngularJS.
- Se puede poner "data-" como prefijo para que estas directivas cumplan la validación HTML5 (ejemplo: data-ng-app).

Módulos

- Forma de organizar el código
- DI(inyección de dependencias)

```
<html ng-app="myModule">
```

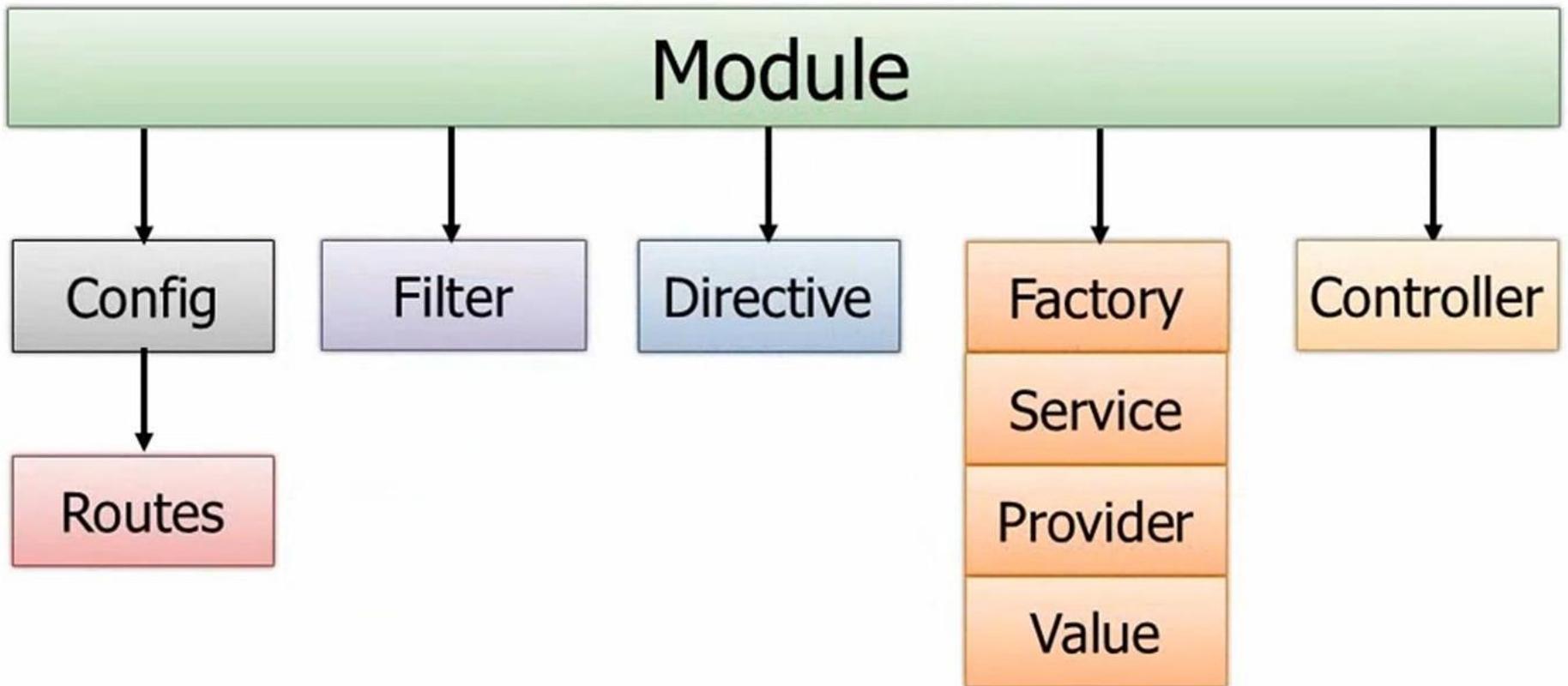
```
// declare a module  
angular.module('myModule', []);
```

Nombre

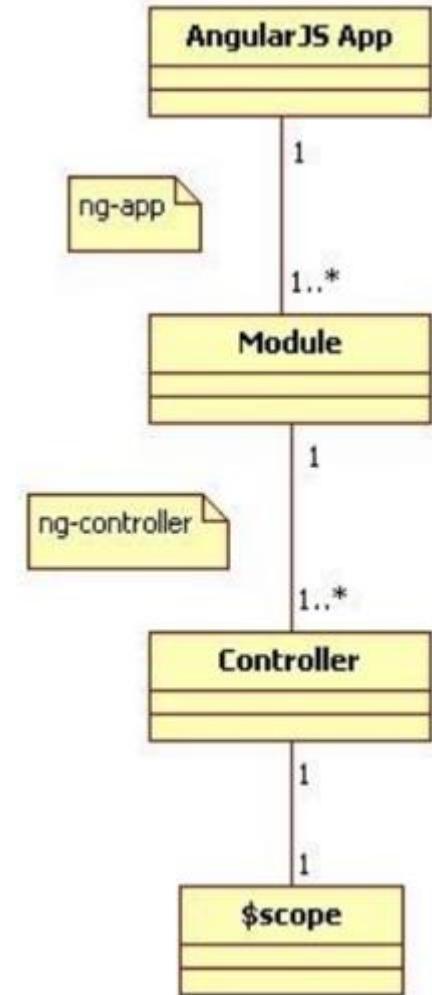
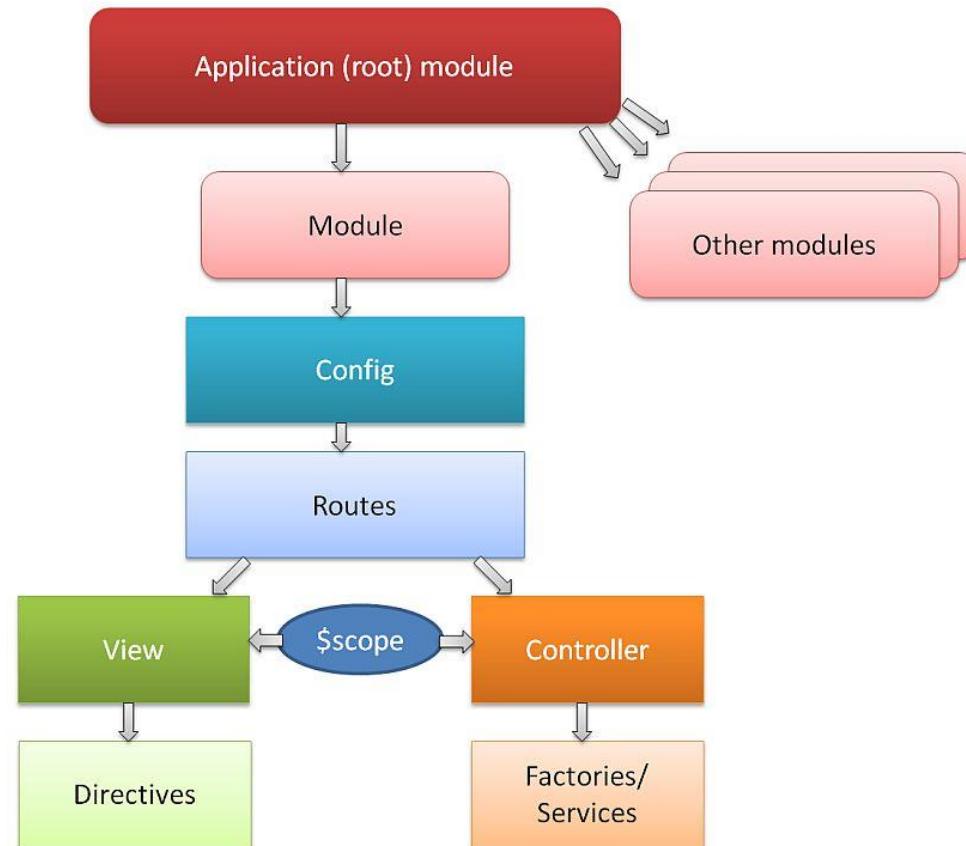
Dependencias

```
angular.module('myModule', ['myOtherModule']);
```

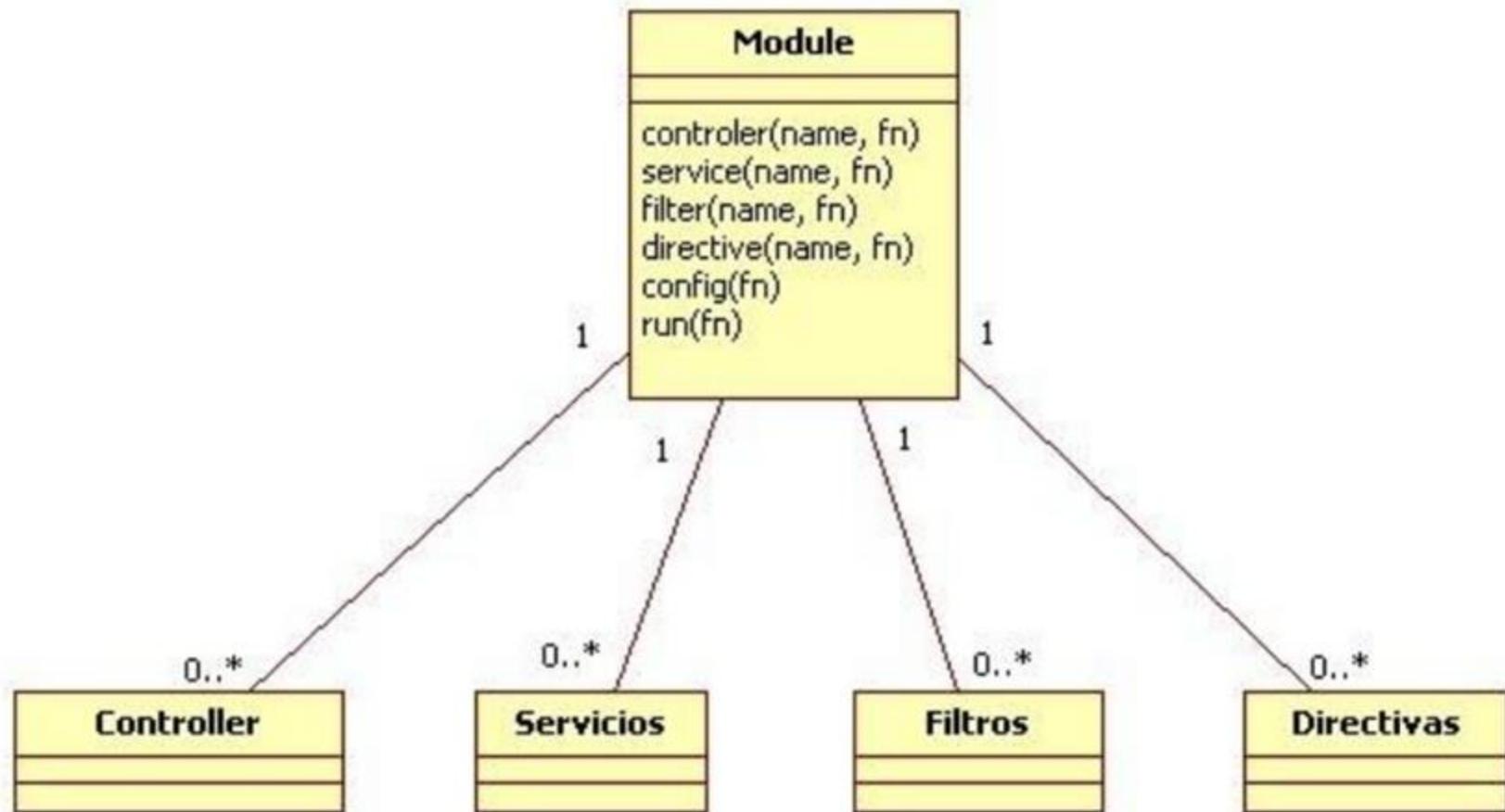
Módulos



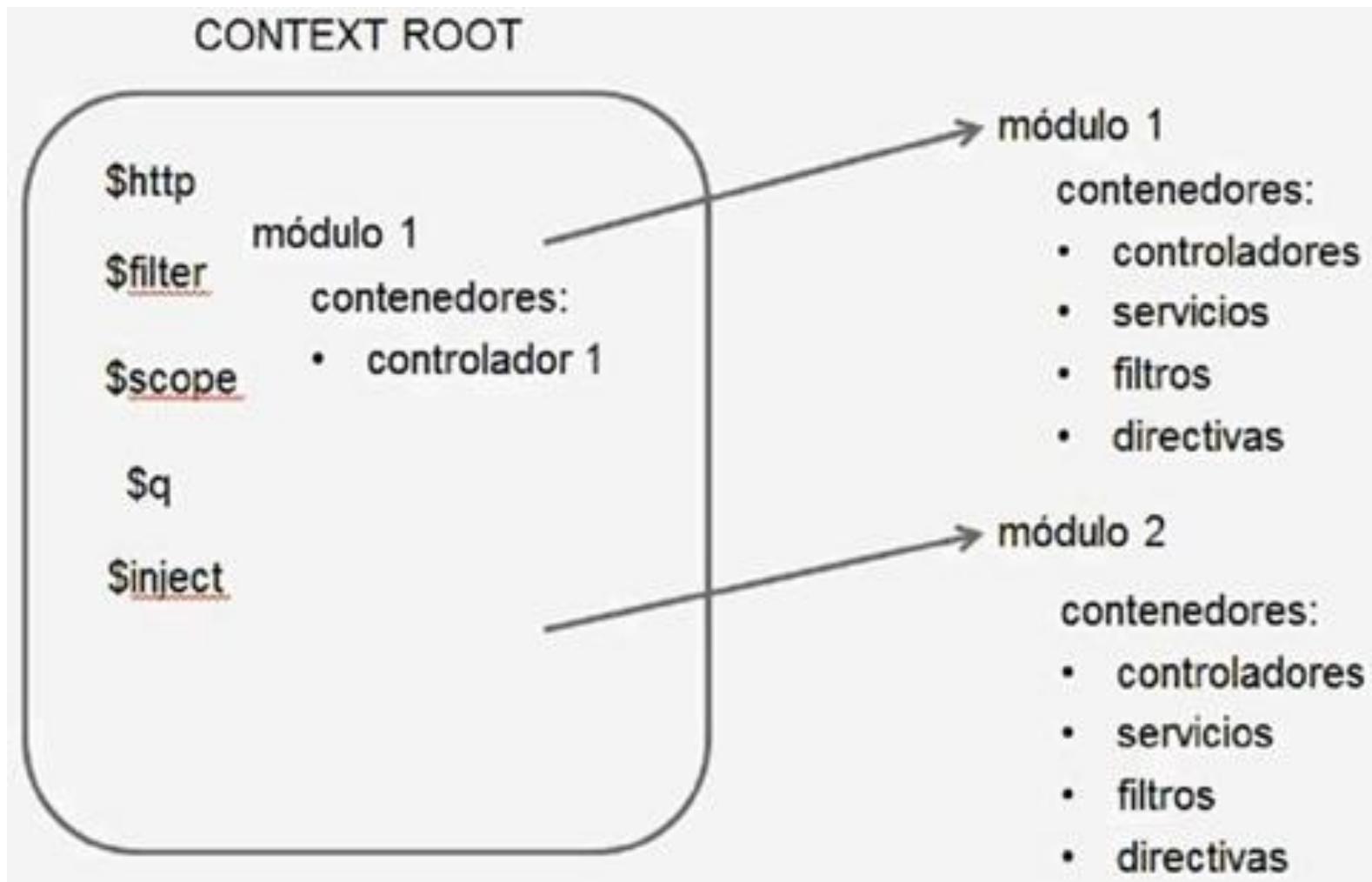
Módulos



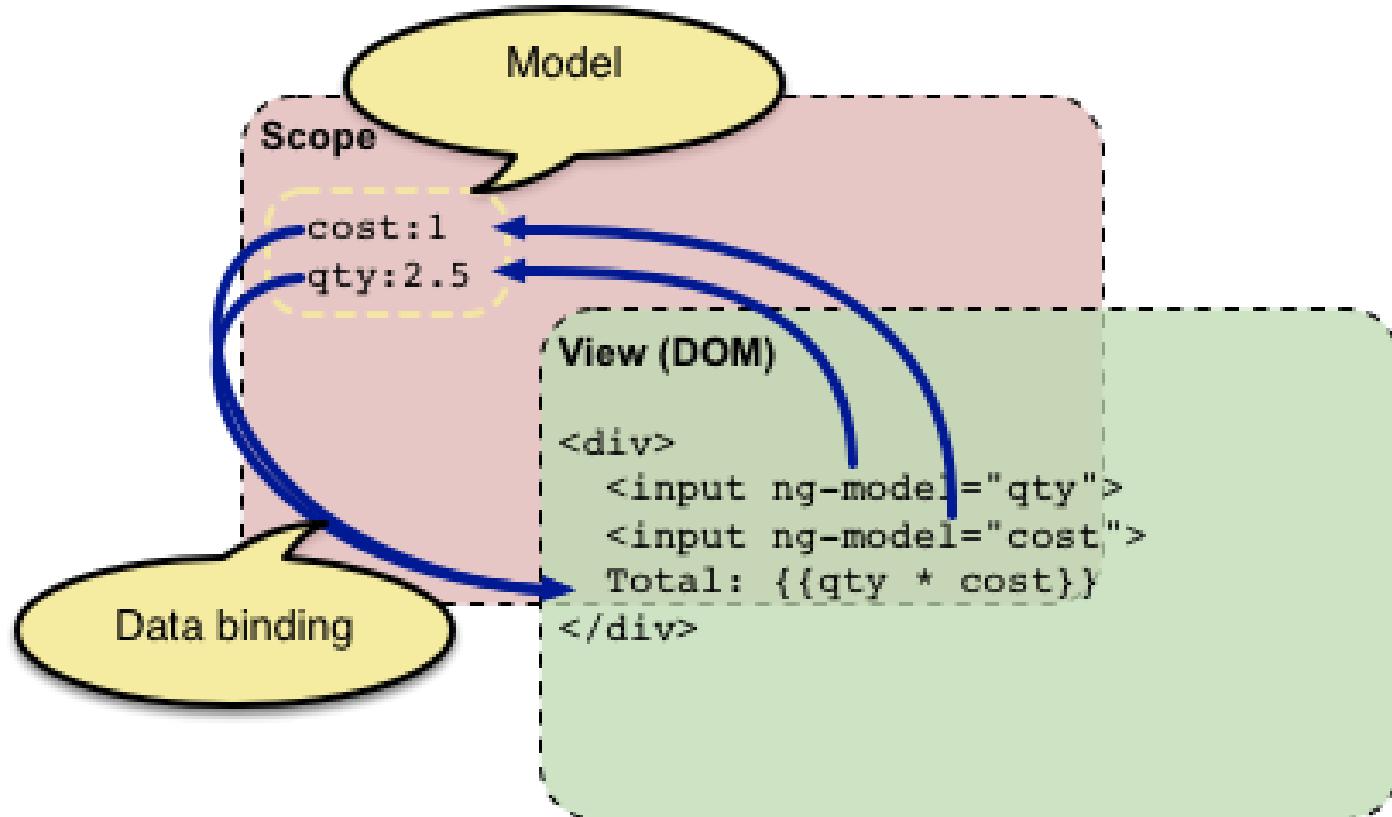
Contenido de un módulo



Módulos

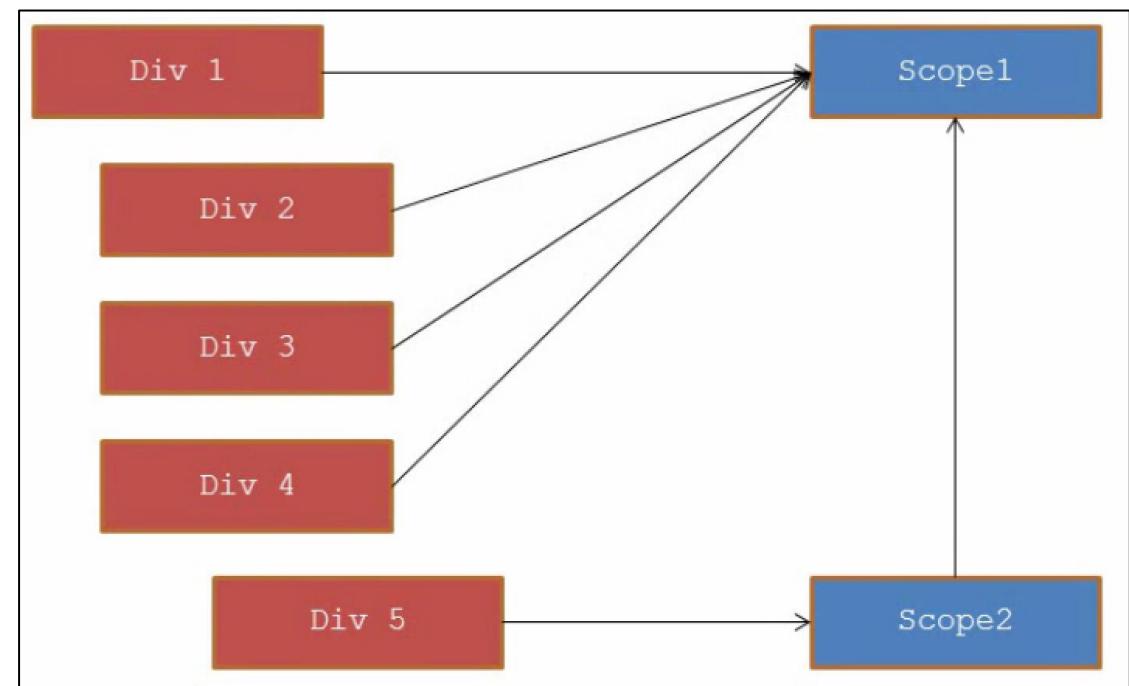


Scope



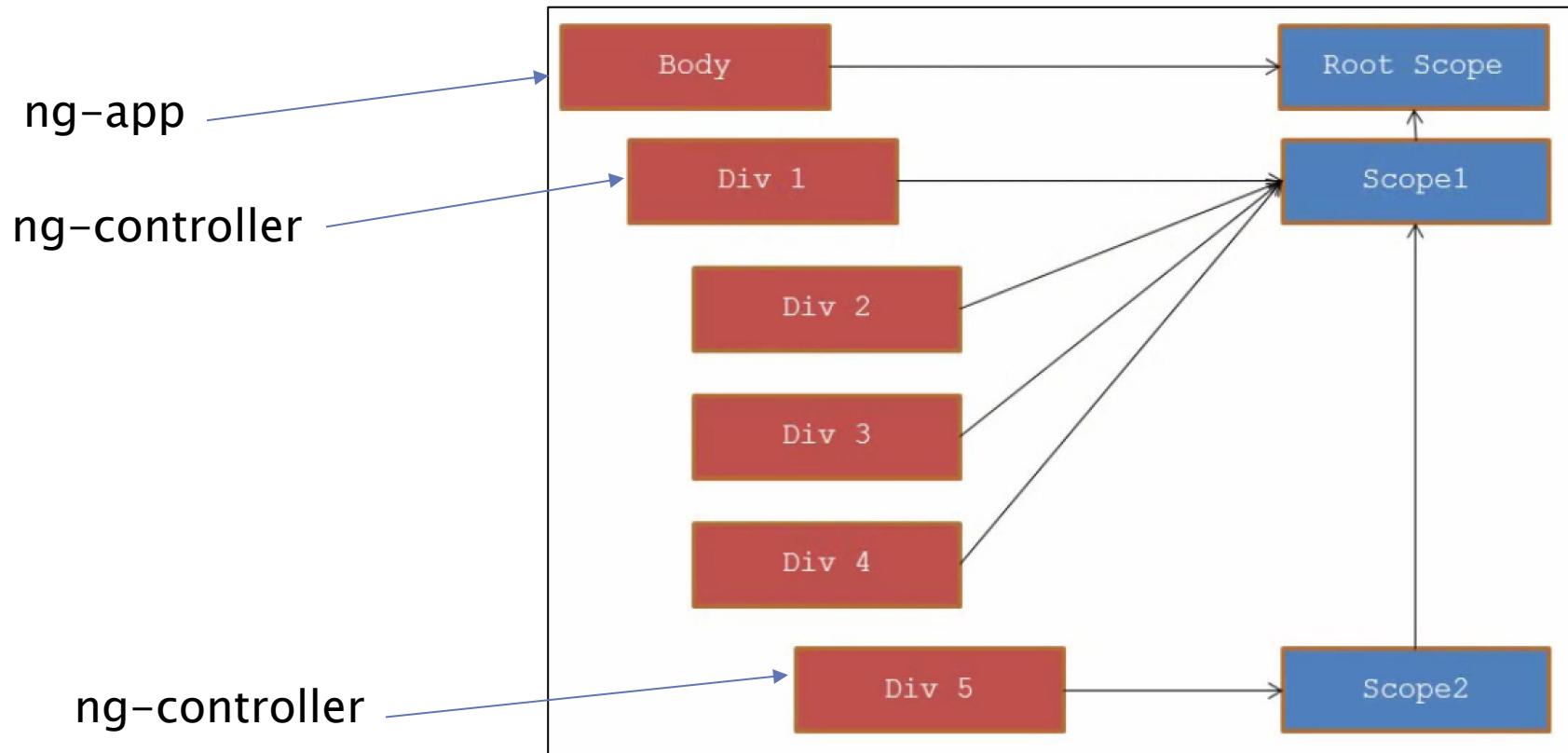
Herencia de Scope

```
<div id="1" ng-controller="myCtrl1">
  <div id="2"></div>
  <div id="3"></div>
  <div id="4" ng-controller="MyCtrl2">
    <div id="5" ></div>
  </div>
</div>
```



Herencia de Scope

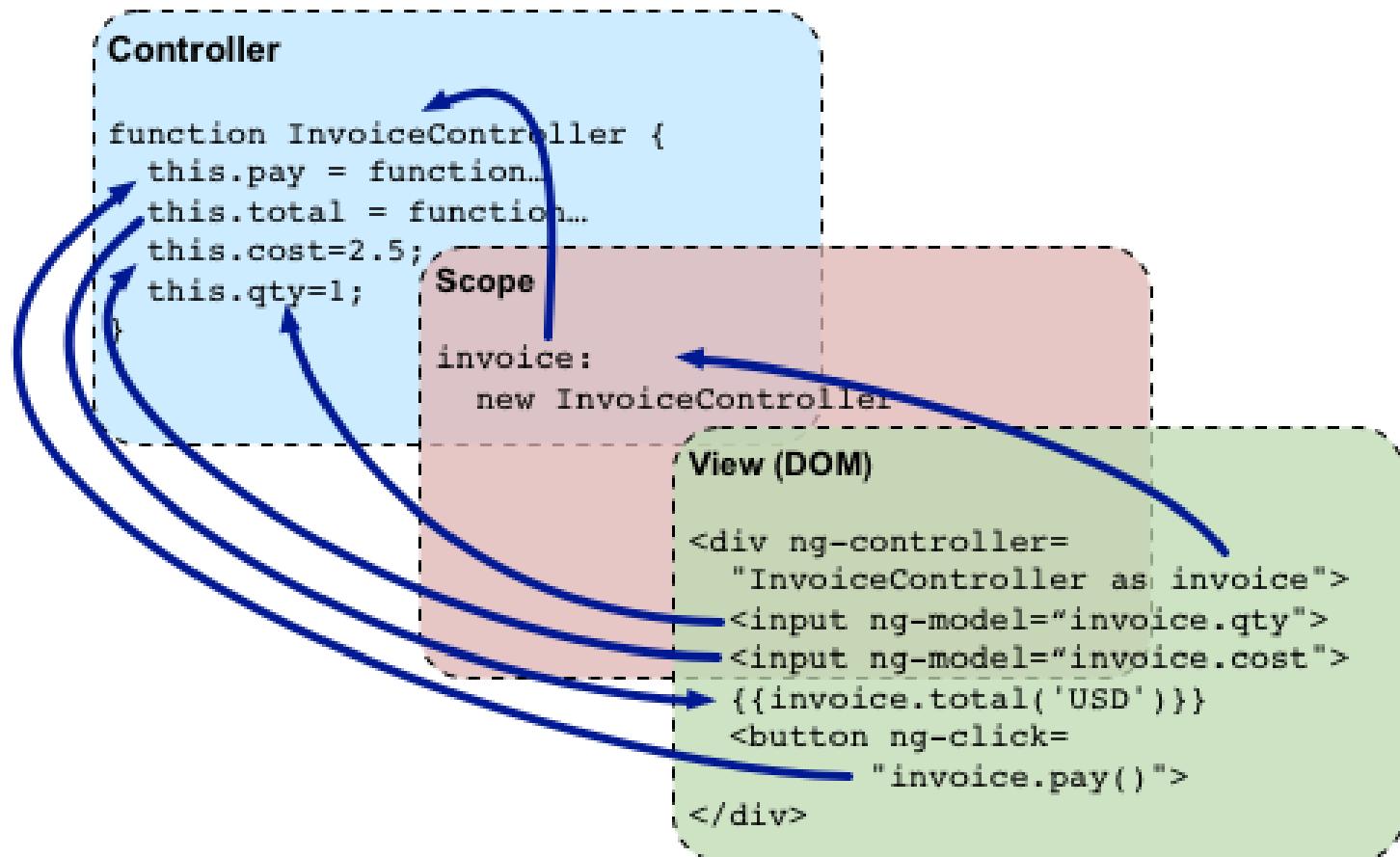
AngularJS siempre va a crear un **rootScope** impulsado por la directiva **ng-app**.



Métodos del scope

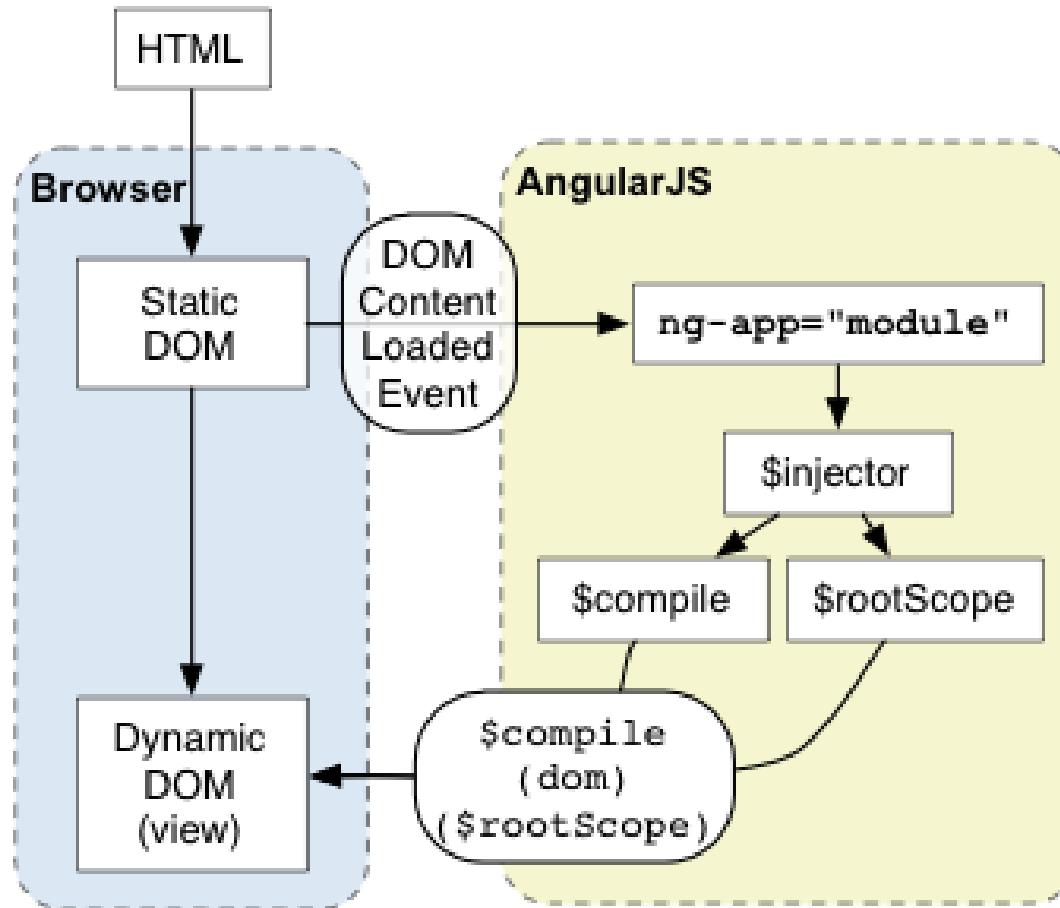
- ▶ \$id
- ▶ \$parent
- ▶ \$new
- ▶ \$apply
- ▶ \$digest
- ▶ \$\$phase
- ▶ \$eval
- ▶ \$evalAsync
- ▶ \$destroy
- ▶ \$watch
- ▶ \$watchGroup
- ▶ \$watchCollection

Controladores



1. El browser carga el html y parsea este dentro del DOM.
2. El browser carga la libreria de angular.js
3. Angular espera por el evento **DOMContentLoaded**.
4. Angular busca la directiva **ng-app**, la cual define el alcance de la aplicación.
5. El módulo especificado en **ng-app** (si hay alguno) es usado para configurar el **\$injector**.
6. El **\$injector** se utiliza para crear el servicio **\$compile**, así como el **\$rootScope**.
7. El servicio **\$compile** es usado para compilar el DOM y linkearlo con el **\$rootScope**.

DOM



Compiler: recorre el DOM y recoger todas las directivas. El resultado es una función linkeada.

Link: combinar las directivas con un **Scope** y produce un **live view**. Cualquier cambio en el **modelo del Scope** se reflejan en la vista, y cualquier interacción del usuario con la vista se reflejan en el **modelo del Scope**. Esto produce **Two Way Data Binding**

Controladores

- Un controlador es una función constructora JavaScript que se utiliza para trabajar con el ámbito de aplicación (modelo de vista).
- Los controladores están asociadas al DOM utilizando directiva ng-controller.

```
angular.module('myModule')
.controller("myController", ["$scope", function($scope){
  //controller body
}]);
```

Nombre

Dependencias

Función constructor

Nuestro primer controlador

- Los datos de las aplicaciones en AngularJS suelen estar controladas por **controllers**.
- Un controlador es un **objeto JS** que gestiona el ámbito de las variables (\$scope).
- Cuando usamos la directiva **ng-controller**, AngularJS llama a ese controlador pasándole el \$scope.

Nuestro primer controlador

```
<script>
var app = angular.module("myApp", []);
app.controller("usuarioCtrl", function ($scope) {
$scope.usuarios = [
{ nombre: 'admin', password: 'admin' },
{ nombre: 'user', password: 'user' },
{ nombre: 'root', password: 'root' }
];
$scope.nombreCompleto = function (u) {
return u.nombre + ' ' + u.password;
};
});
</script>
```

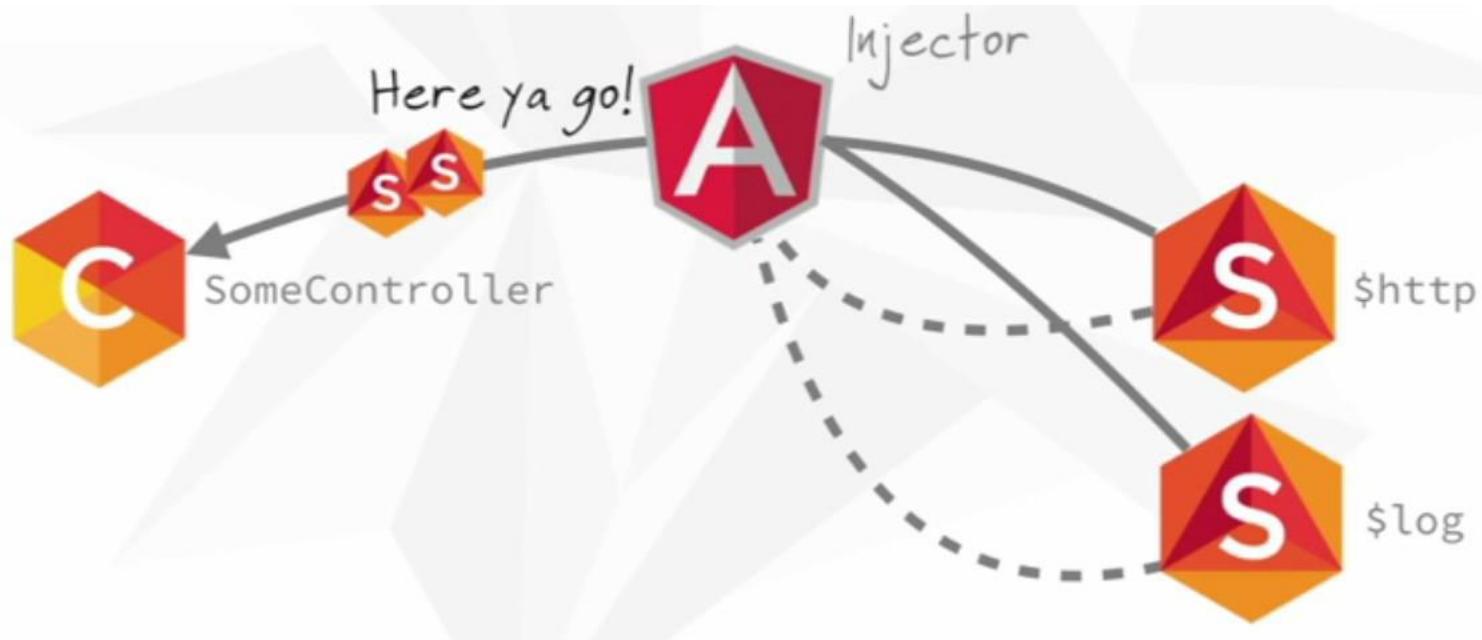
Controladores en ficheros separados

- **app.js**
- **var app = angular.module("myApp", []);**
- **usuarioCtrl.js**
- **index.html**
- **<script src="app.js"></script>**
- **<script src="usuarioCtrl.js"></script>**
- **<div ng-controller="usuarioCtrl">**

Inyección de dependencias

- Las dos principales funcionalidades que se usan en AngularJS son:
- El “**double-binding**” (enlazado doble).
- La **inyección de dependencias**.
- **Puede injectarse servicio en el controlador, directiva o filtro.**
- **Lo que no se puede es injectar un controlador dentro de otro controlador**

Inyección de dependencias



Inyección de dependencias \$inject

```
controller.$inject=['$scope'];

function controller($scope){

    //lógica del controlador

}
```

Separacion de modelo de datos

```
<input type="text" ng-model="model.name">
<input type="text" ng-model="model.surname">
<span>{{fullname()}}</span>

function controller($scope) {
  $scope.fullname = function() {
    return (model.name || "") + " " + (model.surname || "");
  }
}
```

Expresiones

- Aritméticas: {{ euros * 166.386 }}
- Concatenar strings: {{ usuario.nombre + '' + usuario.apellido }}
- Pueden usarse literales, operadores y variables(arrays, objetos).
- No pueden usarse **bucles, condicionales** o **excepciones** (pero sí el operador ternario).

Tipos de datos

- **input con type**
 - text : String
 - number : number
 - checkbox : boolean
- **No existe el tipo Date (No existe type='date')**

Tipos de datos

```
$scope.typeof=function(dato) {  
    return typeof(dato);  
}  
}
```

```
$scope.instanceOfDate=function(obj) {  
    return (obj instanceof Date);  
}  
}
```

Observadores/watchers

- la función **\$watch** acepta como parámetros:
 - 1. la cadena que definimos como modelo
 - 2. función que recibe a su vez como parámetros el valor nuevo y el anterior.
 - En las variables nuevo y anterior tenemos lo que el usuario va escribiendo en el campo de texto

Observadores/watchers

Password:<input type="password" ng-model="password">

```
(function(myApp) {
    controller.$inject=["$scope"];
    function controller($scope) {

        $scope.errorPassword =true;

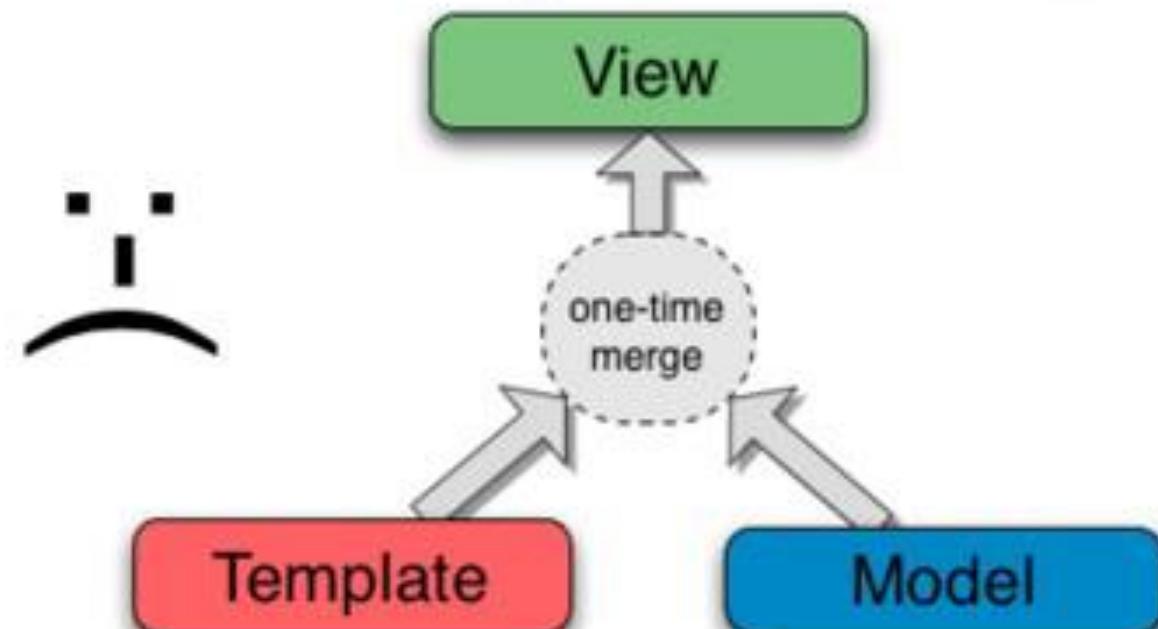
        $scope.$watch('password',function(nuevo,anterior) {
            if(!nuevo) return;
            if(nuevo.length<6) {
                $scope.errorPassword =true;
            }else{
                $scope.errorPassword =false;
            }
        });
    }
    myApp.controller("myController", controller);
})(angular.module("myApp"));
```

Binding

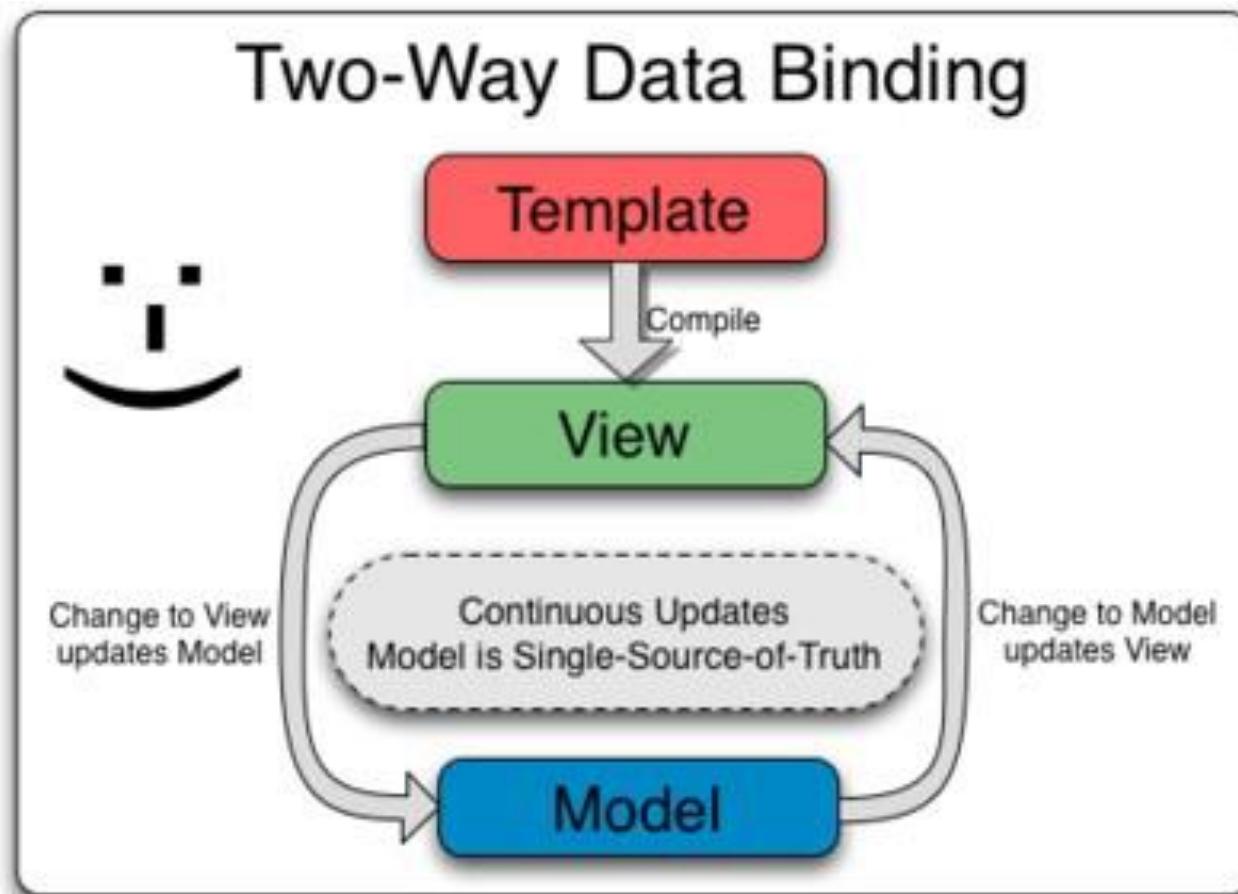
- Enlaza informacion que se quiere mostrar y el aspecto cómo se quiere mostrar
- **One-time binding**
 - Mustache syntax{{::}}
- **One-way binding**
 - ngBind / Mustache syntax{{}}
- **Two-way binding**
 - ngModel

Binding

One-Way Data Binding



Two-way data binding



Two-way data binding

SCRIPT.JS

```
angular.module('scopeExample', [])
.controller('MyController', ['$scope', function($scope) {
    $scope.username = 'World';

    $scope.sayHello = function() {
        $scope.greeting = 'Hello ' + $scope.username + '!';
    };
}]);
```

INDEX.HTML

```
<div ng-controller="MyController">
  Your name:
  <input type="text" ng-model="username">
  <button ng-click='sayHello()>greet</button>
  <hr>
  {{greeting}}
</div>
```

Two-way data binding

```
<!DOCTYPE html>
<html>

  <head>
    <script data-require="angular.js@*" data-semver="1.3.0-beta.5" src="https://code.angularjs.org/1
      <link rel="stylesheet" href="style.css" />
      <script src="script.js"></script>
    </head>

  <body ng-app ng-init="name = 'World'">
    Say hello to: <input type="text" ng-model="name">
    <h1>Hello, {{name}}
```

Binding to model (variable)

Renders model value

Directivas AngularJS

- Las directivas son marcadores en un elemento DOM (como un atributo o un nuevo elemento) que diga AngularJS añadir un comportamiento específico al DOM

ng-app	ng-bind	ng-blur	ng-change
ng-class	ng-click	ng-copy	ng-dblclick
ng-disabled	ng-focus	ng-hide	ng-href
ng-if	ng-include	ng-init	ng-keydown
ng-keypress	ng-keyup	ng-model	ng-mouseenter
ng-mouseleave	ng-mousemove	ng-mouseover	ng-paste
ng-repeat	ng-style	ng-switch	ng-transclude

Directivas AngularJS para eventos

- Teclado:
 - ng-keypress
 - ng-keydown
 - ng-keyup
- Ratón:
 - ng-click
 - ng-dblclick
 - ng-mousedown
 - ng-mouseup
- Formularios
 - ng-change
 - ng-focus

Directivas AngularJS para eventos

```
<div ng-app="" ng-init="count = 0">
<button ng-click="count = count + 1">
Click me!</button>
    <p>{{ count }}</p>
</div>
```

Directivas para manejar el DOM

- **ng-show:** muestra el elemento si la expresión es cierta.
- **ng-hide:** oculta el elemento si la expresión es cierta.
- **ng-disabled:** deshabilita el elemento si la expresión es cierta.
- **ng-if:** elimina del árbol DOM el elemento si la expresión es falsa.

Directiva ng-switch

```
<div ng-switch="mainCtrl.currentTab">
  <div ng-switch-when="tab1">
    Tab 1 is selected
  </div>
  <div ng-switch-when="tab2">
    Tab 2 is selected
  </div>
  <div ng-switch-when="tab3">
    Tab 3 is selected
  </div>
  <div ng-switch-default>
    No known tab selected
  </div>
</div>
</div>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8
<script type="text/javascript">
  angular.module('switchApp', [])
    .controller('MainCtrl', [function() {
      this.currentTab = 'tab1';
    }]);
```

Directiva ng-options

```
<body ng-controller="MainCtrl">
  <p>
    Selecciona un usuario:
    <select ng-model="usuarioSeleccionado"
      ng-options="usuario.nombre + ' ' + usuario.apel for usuario in usuarios" >
      <option value="">-- Selecciona un usuario--</option>
    </select>
  </p>
  <p>
    El usuario seleccionado es: {{usuarioSeleccionado.nombre}} {{usuarioSeleccionado.apel}}
  </p>
</body>
```

Directiva ng-repeat

```
<html ng-app>
<head>
<script src="js/angular.min.js"></script>
</head>
<body>
<div ng-init="usuarios = [ 'admin', 'user', 'root' ]">
<ul ng-repeat="usuario in usuarios">
<li>{{ usuario }}</li>
</ul>
</div>
</body>
</html>
```

- ¿Qué valor tienen \$index, \$first, \$last, \$middle, \$even, \$odd?

Directiva ng-repeat

- [Quizzpot.com](https://quizzpot.com)
<https://quizzpot.com>
- [Html5 Game Devs](https://html5gamedevs.com)
<https://html5gamedevs.com>
- [CSS Tricks](http://css-tricks.com)
<http://css-tricks.com>
- [Bootstrap](http://getbootstrap.com)
<http://getbootstrap.com>
- [Card](http://jessepollak.github.io/card/)
<http://jessepollak.github.io/card/>

Directiva ng-repeat

```
<ul>
  <li ng-repeat="bookmark in bookmarks">
    <p>
      <a href="{{bookmark.url}}>{{bookmark.name}}</a><br/>
      <small>{{bookmark.url}}</small>
    </p>
  </li>
</ul>
```

```
(function(){
  "use strict";

  angular.module('Bookmarks', [
    //dependencies here
  ])

  .controller('MainController', function($scope){
    $scope.name = 'Carl'
    $scope.categories = ['HTML5', 'JavaScript', 'CSS', 'Games'];
    $scope.bookmarks = [
      {id:1,name:'Quizzpot.com',url:'https://quizzpot.com',category:'Java
      {id:2,name:'Html5 Game Devs',url:'https://html5gamedevs.com',cate
      {id:3,name:'CSS Tricks',url:'http://css-tricks.com',category:'CSS'}
      {id:4,name:'Bootstrap',url:'http://getbootstrap.com',category:'CSS'}
      {id:5,name:'Card',url:'http://jessepollak.github.io/card/',category
    ];
  });
})();
```

Bootstrap

<http://angular-ui.github.io/bootstrap>

```
<script src="lib/angular/ui-bootstrap-0.10.0.min.js"></script>
<script src="lib/angular/ui-bootstrap-tpls-0.10.0.min.js"></script>

angular.module('myApp', ['ui.bootstrap']);
```

Filtros

- Un filtro da formato al valor de una expresión para la visualización al usuario
- Se puede utilizar en plantillas de vista, controladores, servicios y directivas
- Puede crear sus propios filtros (en un módulo)
- Filtros predefinidos

Currency

Date

Filter

Json

LimitTo

Lowercase

Number

OrderBy

Uppercase

Filtro date

```
<html ng-app="myApp">
  <head>
    <script src='http://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js'></script>
    <script>
      var app = angular.module("myApp", []);
      app.controller("clockController", function ($scope) {
        $scope.currentTime = new Date();
      });
    </script>
    <title>Welcome to AngularJS</title>
  </head>
  <body>
    <h1>Hello, World.</h1>
    <p ng-controller='clockController'>
      The current time is {{currentTime | date:'h:mm:ss a'}}.
    <p>
  </body>
</html>
```

Creación de Filtros

Usar el filter provider para crear un nuevo filtro en el módulo

Nombre del filtro

```
angular.module('MyFilterModule', []).  
filter('myfilter', function() {  
    return function(input) {  
        ...  
        return output;  
    };  
});
```

Return de la función filtro. El primer parámetro es la entrada.

Creación de Filtros

```
angular.module('greetingModule', []).  
  filter('trim', function() {  
    return function(input) {  
      return input.trim();  
    };  
  });
```

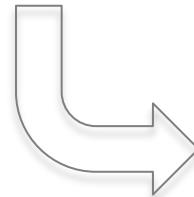
```
<div>{{ text | trim }}</div>
```

Creación de Filtros

```
tempApp.filter('minimum', [function () {
  return function (arrTemp, minimum) {
    var filteredArray = [];
    var min = minimum ? minimum : 15;
    angular.forEach(arrTemp, function (value, key) {
      if (value.temp >= min) filteredArray.push(value);
    });
    return filteredArray;
  };
}]);
```

Filtro reverse

```
9.     <input ng-model="greeting" type="greeting"><br>
10.    No filter: {{greeting}}<br>
11.    Reverse: {{greeting|reverse}}<br>
12.    Reverse + uppercase: {{greeting|reverse:true}}<br>
```



angularjs seminar

No filter: angularjs seminar

Reverse: ranimes sjralugna

Reverse + uppercase: RANIMES SJRALUGNA

Filtro reverse

```
<!doctype html>
<html ng-app="MyReverseModule">
<head>
<script src="http://code.angularjs.org/1.5.8/angular.min.js">
</script>
<script src="script.js"></script>
</head>
<body>
<div ng-controller="reverseController">
<input ng-model="text" type="text"><br>
No filter: {{text}}<br>
Reverse: {{text|reverse}}<br>
</div>
</body>
</html>
```

Filtro reverse

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

myApp.filter('reverse', function() {
  return function(input) {
    var out = '';
    for (var i = 0; i < input.length; i++) {
      out = input.charAt(i) + out;
    }
    return out;
  }
});
```

Filtro en ngRepeat

```
<html ng-app>
  <body>
    <div ng-init="libros =
    [
      {titulo:'El Juego de Ender', autor:'Orson Scott Card'},
      {titulo:'Juego de Tronos', autor:'George R.R. Martin'},
      {titulo:'I robot', autor:'Isaac Asimov'}
    ]">

      <h1>Listado de libros</h1>
      Nombre: <input type="text" ng-model="nombre" />

      <ul>
        <li ng-repeat="libro in libros | filter:nombre | orderBy:'autor'">{{ libro.titulo |
uppercase }} - {{ libro.autor}}</li>
      </ul>
    </div>
    <script src="angular.min.js"></script>
  </body>
</html>
```

Routing

- La navegación entre páginas se proporcionada por el módulo de **ngRoute**
- Las rutas se declaran a través de la **\$routeProvider** del servicio \$route
- Es compatible con enlaces tipo historial, marcadores y botones Adelante / Atrás.
- Vistas parciales gracias a la directiva **ngView**
- <https://docs.angularjs.org/api/ngRoute>

ngRoute

ng	module
	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	

ngRoute

The ngRoute module provides routing and deeplinking services and directives for angular apps.

Example

See [\\$route](#) for an example of configuring and using ngRoute.

Installation

First include angular-route.js in your HTML:

```
<script src="angular.js">
<script src="angular-route.js">
```

You can download this file from the following places:

- [Google CDN](#)
e.g. "<http://ajax.googleapis.com/ajax/libs/angularjs/X.Y.Z/angular-route.js>"
- [Bower](#)
e.g. `bower install angular-route@X.Y.Z`
- [code.angularjs.org](#)
e.g. "<http://code.angularjs.org/X.Y.Z/angular-route.js>"

AngularJS version you are running.

Include ngRoute in your application by adding it as a dependent module:

```
angular.module('app', ['ngRoute']);
```

ngRoute

- Proporciona servicios de deep linking y navegación entre pantallas.
- Hay que incluir angular-route.js en index.html como librería independiente de la de angular.js
- Es necesario marcar ngRoute como un módulo dependiente.

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

index.html

```
<div>
```

```
    <ng-view></ng-view>
```

```
</div>
```

ngRoute

```
<!doctype html>
<html>
  <head>
    <title>My Angular App</title>
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.js">
    </script>
  </head>
  <body ng-app="myApp">
```

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

app.config(['$routeProvider',function($routeProvider){
  // Your code here
}]);
```

Routing

```
var myApp = angular.module('myApp', ['ngRoute']);
myApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/phones', {
        templateUrl: 'partials/phone-list.html',
        controller: 'PhoneListCtrl'
      }).
      when('/phones/:phoneId', {
        templateUrl: 'partials/phone-detail.html',
        controller: 'PhoneDetailCtrl'
      }).
      otherwise({
        redirectTo: '/phones'
      });
}]);
```

Módulo ngRoute

Route

Ruta por defecto

Routing

```
(function(myApp) {
  config.$inject = ['$routeProvider'];
  function config($routeProvider) {
    $routeProvider
      .when("/", {templateUrl:'list.html'})
      .when("/edit/:id", {templateUrl:'edit.html'})
      .when("/create", {templateUrl:'create.html'})
      .when("/delete/:id", {templateUrl:'delete.html'})
      .otherwise({redirectTo:'/'});
  }
  myApp.config(config);
})(angular.module("myApp"));
```

Routing Habilitar HTML5

```
.config(['$routeProvider', '$locationProvider',
function($routeProvider, $locationProvider) {
    // use the HTML5 History API
    $locationProvider.html5Mode({
        enabled: true,
        requireBase: false
    });
}]);
```

Get Router parameters

PhoneDetailCtrl

```
function PhoneDetailCtrl($scope,  
$routeParams) {  
  
  $scope.phonelId = $routeParams.phonelId;  
  
}
```

Routing

```
myApp.config(['$routeProvider', function ($routeProvider) {
  $routeProvider
    .when('/', {
      templateUrl: 'views/main.html'
    })
    .when('/cart/item:id', {
      templateUrl: 'views/cart-item.html'
    })
    .otherwise({
      redirectTo: '/'
    });
}]);
```

Ejercicio Routing

[Listado libros](#) [Añadir libro](#)

Numero libros eliminados: 1 Numero libros nuevos: 1

Buscador

Titulo:

Autor:

búsqueda exacta

Ordenar por

- [1. El juego de Ender - Orson Scott Card \[eliminar\]](#)
- [2. I robot - Isaac Asimov \[eliminar\]](#)
- [3. Juego de Tronos - George R.R. Martin \[eliminar\]](#)

Formularios /Validación

```
<body ng-app>
  <form name="form">
    <input type="text" name="name" ng-model="name" required/>
    <span ng-show="form.name.$error.required">name requerido</span>
  </form>

  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
</body>
```



Validaciones

- required de HTML5
- ng-required="true"
- min
- max
- ng-minlength
- ng-maxlength
- ng-pattern

ng messages

```
<script src="https://code.angularjs.org/1.5.8/angular-messages.js">
</script>
```

```
<form name="form" ng-submit="enviar()" ng-controller="myController">
<input type="text" name="name" ng-model="model.name" ng-
required="true" ng-minlength="5" ng-maxlength="10" />
<div ng-messages="form.name.$error">
<div ng-message="required">Campo requerido</div>
<div ng-message="minlength">La longitud del campo no puede ser
menor que 5</div>
<div ng-message="maxlength">La longitud del campo no puede ser
mayor de 10</div>
</div>
<button type="submit">Enviar</button>
</form>
```

Formulario validaciones

nombre:

apellidos:

email: Email incorrecto

website:

provincia: -- Seleccione una opción -- ▾

suscribirse a la newsletter

Registrar

```
{  
  "name": "name",  
  "lastName": "surname surname2"  
}
```

nombre:

apellidos:

email:

website: Formato de URL incorrecto

provincia: -- Seleccione una opción -- ▾

suscribirse a la newsletter

Registrar

```
{  
  "name": "name",  
  "lastName": "surname surname2",  
  "email": "user@domain.com"  
}
```

Directivas custom

- Directiva personalizada puede ser utilizada para proporcionar elementos html de forma declarativa y reutilizables.
- Hay muchas formas de declarar una directiva en el marcado usando (atributos HTML, los elementos, los comentarios y las clases CSS).
- Es una buena práctica poner prefijo a los nombres de las directivas para evitar la confusión con otros elementos HTML

```
<my-dir></my-dir>
<span my-dir="exp"></span>
<!-- directive: my-dir exp -->
<span class="my-dir: exp;"></span>
```

Directivas custom

- Permiten extender el HTML
- Podemos restringir el uso de una directiva para un contexto específico:
- ‘E’ = Elemento <my-directive></my-directive>
- ‘A’ = Atributo <div my-directive="exp"> </div>
- ‘C’ = Class <div class="my-directive: exp;"></div>
- ‘M’ = Comentario

Directivas custom

```
myapp.directive('mydirective',function($inject){  
  return {  
    DDO:Directive Definition Object  
  };  
})
```

Templates en directivas custom

```
.directive('sampleDirective', function(){
return {
    link: function(scope, element, attrs) {
    }
},
restrict: 'E',
template: '<div>Hello, World!</div>'
//or:
templateUrl: 'path/to/file.html'
});
```

Directivas custom

```
angular.module('greetingModule', []).  
directive('helloWorld', function() {  
    return {  
        restrict: 'A',  
        scope: true,  
        link: function(scope, element, attrs) {  
            scope.greeting = 'Hello world!';  
        },  
        template: '<div>{{ greeting }}</div>'  
    };  
});
```

```
<div hello-world></div>
```

Directivas/Paso de atributos

```
<body ng-app="myApp">
  <my-directive data-language="es"></my-directive>
  <my-directive data-language="en"></my-directive>

  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
  <script src="myApp.js"></script>
  <script src="myDirective.js"></script>
</body>
```

Directivas/Paso de atributos

```
(function(myApp) {
    function directive() {
        return {
            restrict:'E',
            template:function(element,attrs){
                if(attrs.language=='es'){
                    return '<div>Hola mundo</div>';
                }else{
                    return '<div>Hello world</div>';
                }
            },
            replace:true
        };
    }
    myApp.directive("myDirective", directive);
})(angular.module("myApp"));
```

Zip Code Input

Zips are allowed in one of the following formats

- 12345
- 12345 1234
- 12345-1234

Enter valid zip code:

Directivas/Custom validator

DNI/NIF:

11222333N

DNI/NIF válido

Submit

DNI/NIF:

11222333

DNI/NIF inválido

Submit

ToDo list

Search for:

New Task:

Task 1

Task 2

Directivas/Product List

Products

Name	Category	Price	
Dummy1	Test	\$1.25	<button>Delete</button> <button>Edit</button> <button>+</button>
Dummy2	Test	\$2.45	<button>Delete</button> <button>Edit</button> <button>+</button>
Dummy3	Test	\$4.25	<button>Delete</button> <button>Edit</button> <button>+</button>

RefreshNew

Servicios

- Componentes reutilizables de lógica de negocio, independientemente de puntos de vista, conectados entre sí mediante la inyección de dependencias (DI).
- Objectos singleton generados por una factory.
- Servicios predefinidos y que podemos injectar en nuestros controladores:
 - **\$log, \$http, \$filter, \$exceptionHandler**

Angular Services



- angular comes with a bunch of services built-in to give more features to controller
- examples like fetching json from web service using the \$http service, logging messages to javascript console \$log, filter in array \$filter service
- you notice something here all the services begin with \$ sign, that's because all of these angular built-in functions

How to use service

service as argument

```
app.controller('lesson12',['$http',function($http){  
    service name  
}]);
```

User more services

```
app.controller('lesson12',['$http' , '$log' , function($http , $log){  
    service as argument  
}]);
```

Servicios predefinidos

\$rootScope	Padre de todos los scopes
\$location	Acceso a una URL
\$routeProvider	Navegación
\$http	Peticiones HTTP
\$resource	Peticiones API REST
\$log	console.log()

Servicios \$log

- Servicio equivale al console.log()
- Proporciona una serie de métodos por defecto en función del nivel del log que quieras mostrar.
 - \$log.log("log")
 - \$log.info("info")
 - \$log.warn("warning")
 - \$log.error("Error")

```
app.controller('SomeController', ['$log', function($log){}]);
```

Servicios

```
var countryApp = angular.module('countryApp', []);
countryApp.controller('CountryCtrl', ['$scope', '$http', function
(scope, http) {
    http.get('countries.json').success(function(data) {
        scope.countries = data;
    });
}]);
```

Injectar servicio \$http
mediante DI

Pasar el objeto http
como parámetro

Llamada al método
GET

Creando servicios

Registrar nueva función factory

```
var myModule = angular.module('myModule', []);
myModule.factory('serviceId', function() {
    var shinyNewServiceInstance;
    //factory function body that constructs
shinyNewServiceInstance
    return shinyNewServiceInstance;
});
```

Devolver una
instancia del servicio

Servicio \$http

- El servicio básico para hacer todas las peticiones HTTP mediante AJAX
- Proporciona una serie de métodos por defecto
 - `$http.post(url, config)`
 - `$http.get(url, config)`
 - `$http.put(url, config)`
 - `$http.delete(url, config)`
- Utiliza la API de promesas para devolver el resultado
 - Proporcionado por el servicio \$q

Servicio http

- El servicio \$http es la forma en que realizamos una solicitud asíncrona a un servidor
- Usando \$http como una función con un objeto de opciones.
- La llamada devuelve una promesa con métodos .success () y .error ()
- Si le decimos a \$http que obtenga JSON, el resultado será automáticamente decodificado en objeto JavaScript.

```
1  $http({method: 'GET', url: '/someUrl'}).
2    success(function(data, status, headers, config) {
3      // this callback will be called asynchronously
4      // when the response is available
5    }).
6    error(function(data, status, headers, config) {
7      // called asynchronously if an error occurs
8      // or server returns response with an error status.
9    });

```

Ejemplo: inyección del servicio \$http

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
$http.get("http://www.w3schools.com/angular/customers.php")
.success(function (response) {
$scope.names = response.records;
});
});
```

Servicio http

```
(function(){
  var app = angular.module('store', [ 'store-products' ]);

  app.controller('StoreController', [ '$http',function($http){
    var store = this;
    store.products = [ ];

    $http.get('/products.json').success(function(data){
      store.products = data;
    });
  }]);
})();
```

Promesas

```
$http.post('/user/', user).then(  
  function() {  
    alert('Successfully created user!');  
  }, function() {  
    alert('Error creating user...');  
  }  
);
```

Servicio Custom

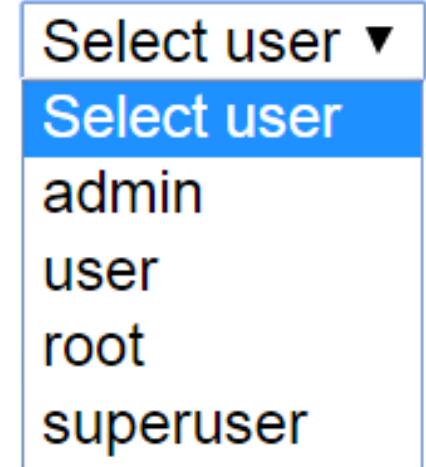
```
var app = angular.module('myApp', []);
app.factory('testFactory', function(){
return {
sayHello: function(text){
return "Factory says \"Hello " + text + "\"";
},
sayGoodbye: function(text){
return "Factory says \"Goodbye " + text + "\"";
}
};
});

function HelloCtrl($scope, testService, testFactory)
{
$scope.fromFactory = testFactory.sayHello("World");
}
```

```
<div ng-controller="HelloCtrl">
<p>{{fromFactory}}</p>
</div>
```

Cargar json mediante Servicio

```
[  
{"id": "1", "name": "admin"},  
 {"id": "2", "name": "user"},  
 {"id": "3", "name": "root"},  
 {"id": "4", "name": "superuser"}  
]
```



```
<body ng-app="myApp" ng-controller="myController">  
  <select ng-model="model.name" ng-options="item.name as item.name for item in collection"  
    ng-change="changeUser(model)"></select>  
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>  
  <script src="myApp.js"></script>  
  <script src="myController.js"></script>  
  <script src="myService.js"></script>
```

Cargar json mediante Servicio

```
(function(myApp) {  
    controller.$inject=["myService","$scope"];  
    function controller(myService,$scope) {  
  
        myService.getData().success(function(data) {  
            $scope.collection=data;  
            $scope.collection.splice(0,0,{"id":"0","name":"Select user"});  
            $scope.model=$scope.model||{};  
            $scope.model.name="Select user";  
        });  
  
        $scope.changeUser= function(user) {  
            console.log(user);  
        };  
  
    }  
    myApp.controller("myController", controller);  
}(angular.module("myApp")));
```

Cargar json mediante Servicio

```
(function(myApp) {  
    service.$inject = ['$http'];  
    function service($http) {  
        this.getData = function(){  
            return $http.get('data.json');  
        }  
    }  
    myApp.service("myService", service);  
}(angular.module("myApp")));
```

Petición GET

```
angular
  .module('httpModule', [])
  .controller('MainCtrl', function($scope, $http){
    $http.get(
      'http://jsonplaceholder.typicode.com/posts/1/comments',{}
    )
    .success(function(data){
      $scope.resultdata = data;
    })
    .error(function(data){
      alert('Se ha producido un error')
    });
  });
});
```

Petición POST

```
angular
.module('httpModule', [])
.controller('MainCtrl', function($scope, $http,$log){
$http.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded;charset=utf-8';
$http.post(
  'http://jsonplaceholder.typicode.com/users',
  {
    "name": " name",
    "username": "username",
    "email": "user@domain.com",
    "phone": "123456",
    "website": "http://angular.es"
  }
)
.success(function(data){
  $log.info(data)
  $scope.id = data.id;
})
.error(function(error){
  alert('Se ha producido un error'+error);
  $log.info(error)
});
});
```

Petición PUT

```
angular
  .module('httpModule', [])
  .controller('MainCtrl', function($scope, $http){
    $http.put(
      'http://jsonplaceholder.typicode.com/users/10',
      {
        "name": "name",
        "username": "username",
        "email": "user@domain.es",
        "address": {
          "street": "address",
        },
        "phone": "123456",
        "website": "http://angular.es"
      }
    )
    .success(function(data){
      $scope.data = data;
    })
    .error(function(data){
      alert('Se ha producido un error')
    });
});
```

Llamadas servicio REST

- **ngResource**
- El módulo ngResource proporciona soporte de interacción con servicios RESTful a través del servicio \$resource.

```
<script src="angular.js">
<script src="angular-resource.js">
```

Llamadas servicio REST

- **\$resource**
 - Crea un objeto de recurso que le permite interactuar con fuentes de datos RESTful del lado del servidor.
 - [https://docs.angularjs.org/api/ngResource/service/\\$resource](https://docs.angularjs.org/api/ngResource/service/$resource)
-
- var User = \$resource('/user/:userId', {userId:'@id'});
var user = User.get({userId:123});
// GET: /user/123

Llamadas servicio REST

```
{      'get' : {method: 'GET' },
      'save' : {method: 'POST' },
      'query' : {method: 'GET' , isArray:true},
      'remove' : {method: 'DELETE' },
      'delete' : {method: 'DELETE' }
};
```

Llamadas servicio REST

```
angular.module('Bookmarks', [
    //dependencies here
    'ngResource'
])

.service('Category',function($http){
    this.getAll = function(success,failure){
        $http.get('http://bookmarks-angular.herokuapp.com/api/categories')
            .success(success)
            .error(failure);
    }
})

.factory('Bookmark',function($resource){
    return $resource('http://bookmarks-angular.herokuapp.com/api/bookmarks/:id',{
        id : '@id'
    },{
        update : {method:'PUT'}
    });
})
```

Directivas vs Componentes

<https://code.angularjs.org/1.5.8/docs/guide/component>

Comparison between Directive definition and Component definition

	Directive	Component
bindings	No	Yes (binds to controller)
bindToController	Yes (default: false)	No (use bindings instead)
compile function	Yes	No
controller	Yes	Yes (default <code>function() {}</code>)
controllerAs	Yes (default: false)	Yes (default: <code>\$ctrl</code>)
link functions	Yes	No
multiElement	Yes	No
priority	Yes	No
require	Yes	Yes
restrict	Yes	No (restricted to elements only)
scope	Yes (default: false)	No (scope is always isolate)
template	Yes	Yes, injectable
templateNamespace	Yes	No
templateUrl	Yes	Yes, injectable

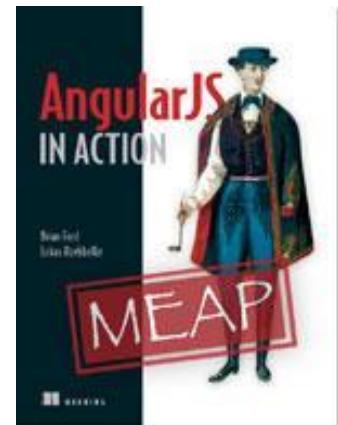
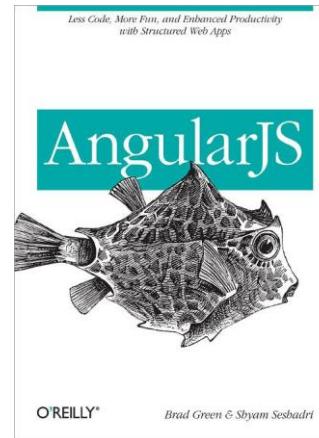
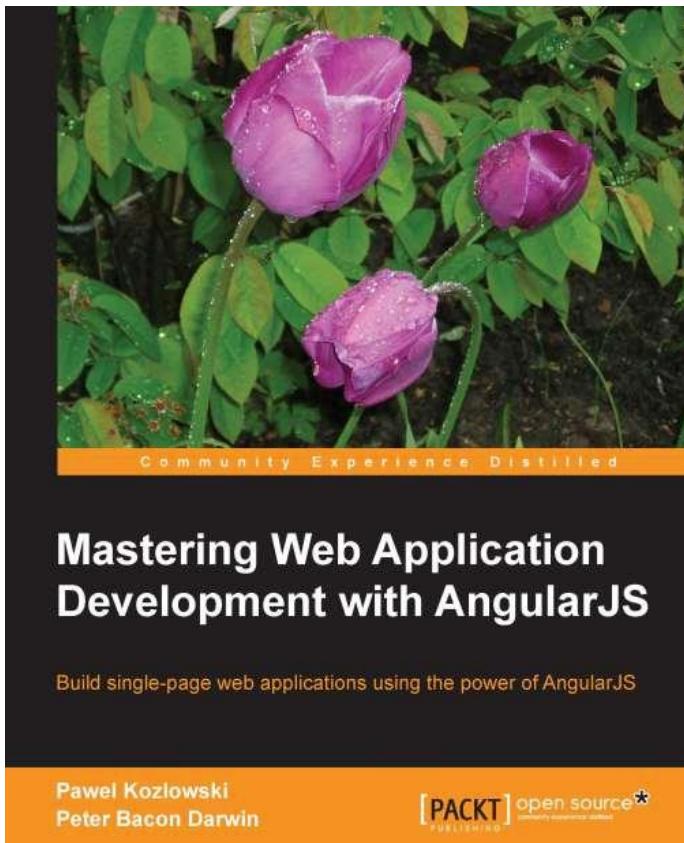
Referencias

- <https://angularjs.org/>
- <https://docs.angularjs.org/guide>
- <https://docs.angularjs.org/tutorial>
- <https://builtwith.angularjs.org>
- <https://egghead.io/technologies/angularjs>

Referencias

- <https://openwebinars.net/tutorial-angularjs/>
- <http://cursoangularjs.es/doku.php>
- <http://www.w3schools.com/angular/>
- <http://raulexposito.com/documentos/como-aprender-angularjs/>
- <https://tombatossals.github.io/angularjs-tutorial/#/>

Libros



Ejercicios

- **ejercicio hola mundo**
- **ejercicio creacion de filtros**
- **ejercicio routing**
- **ejercicio creacion de directivas**
- **ejercicio creacion de servicios**

ejercicio hola mundo

El objetivo de este ejercicio es entender la estructura de un frontal SPI en angular. Para ello, generaremos una aplicación que imprima por pantalla el mensaje “Hola mundo”, el cual, vendrá dado desde un controlador. Mas tarde, modificaremos esa misma aplicación para que imprima el mensaje introducido por un campo de texto. Los conocimientos que pondremos en práctica con este ejercicio son:

1. La estructura de un modulo y un controlador en angular
2. El funcionamiento de la inyección de dependencias en angular
3. El funcionamiento del DOM declarativo de las vistas
4. Los scripts de pruebas unitarias

ejercicio hola mundo

El fichero index.html que creamos cuando preparábamos el entorno de trabajo no esta listo para ejecutar ningún fichero de javascript y, aún menos, nada que tenga que ver con angularjs . Además, angularjs, de por si, no es compatible con navegadores antiguos como internet explorer 8. Para estos caso es necesario añadir librerías extra al index.html que doten al interprete del navegador de las dependencias necesarias para que el framework funcione.

Todo esto se traduce en las siguientes acciones:

- Importar la librería angular.min.js mediante una etiqueta script
- Si el navegador es mas antiguo del internet explorer 9 importar las librerias html5shiv.min.js y respond.min.js

ejercicio hola mundo

Así debe de quedar nuestro fichero index.html:

```
<!DOCTYPE html>
<html>
<head>
<script
src="node_modules/angular/angular.min.js"></script>
<meta charset "UTF-8">
<title>Practicas de angular</title>
</head>
<body>
<h1>Hola mundo</h1>
</body>
</html>
```

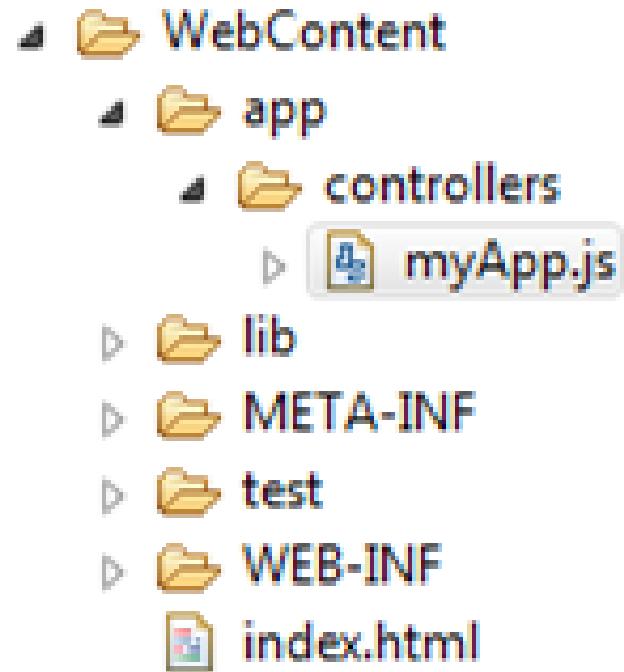


- **Framework angular:** Esta línea carga las librerías core de angular

ejercicio hola mundo

creamos el controlador

Antes de crear el controlador, dentro del directorio WebContent, creamos las carpetas app/controllers donde almacenaremos todos los controladores que vayamos creando en los diferentes ejercicios. El fichero javascript de nuestro primer controlador lo llamaremos `myApp.js`

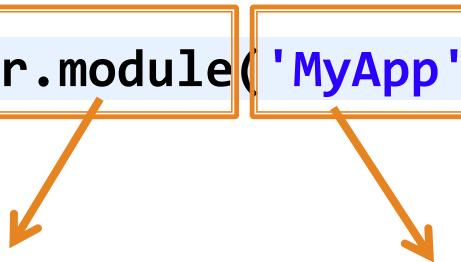


ejercicio hola mundo

Creamos el controlador

Esta es la primera línea de nuestro controlador:

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



- **angular.module**: Esta instrucción genera un nuevo módulo en angular
- **MyApp**: Esta cadena de caracteres es la referencia al nuevo módulo. A la hora de realizar la inyección en root, nos referiremos a los diferentes módulos a través de estas cadenas.

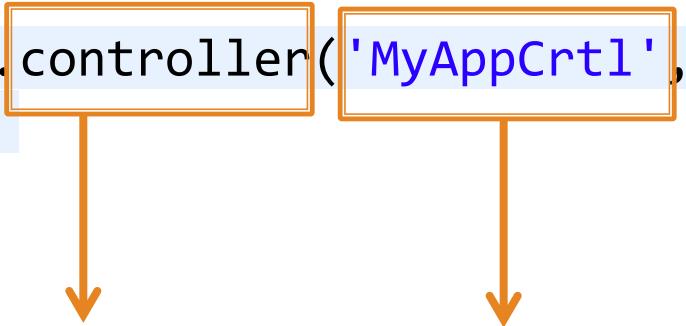
El resultado de la creación del módulo lo almacenamos en la variable `myApp`.

ejercicio hola mundo

creamos el controlador

Definición de un controlador

```
myApp.controller('MyAppCrtl', ['$scope', function($scope){  
    ...  
}])
```



- **controller:** Esta instrucción declara un contenedor de tipo controller
- **MyAppCrtl:** Esta cadena de caracteres es la referencia al nuevo controlador.

ejercicio hola mundo

Creamos el controlador

Definición de un controlador

```
myApp.controller('MyAppCtrl', ['$scope', function($scope){  
    ...  
}])
```

El último parámetro de la definición de un controlador es el array de inyecciones. En él se lista las referencias de otros contenedores que el controlador necesita. Estos contenedores serán inyectados en el módulo root al mismo tiempo que se inyecta nuestro controlador. El último elemento del array es la función que implementa el controlador en sí. Dicha función recibe como parámetro un número igual de variables al número de contenedores inyectados en el controlador. Esto es así porque, angular, asigna el código de los contenedores a las variables de la función en el mismo orden en el que son inyectados. No es necesario que los parámetros de entrada del controlador se llamen igual que los contenedores pero es una buena práctica para evitar confusiones.

ejercicio hola mundo

Creamos el controlador

Código del controlador

```
$scope.message = "Hola Mundo!!!";
```

\$scope es un servicio del propio angular que publica objetos javascript para que puedan ser usados en las vistas. Todo aquello que pongamos dentro del \$scope lo podremos usar en una vista.

En la línea de código de esta transparencia estamos declarando una variable en el scope llamada message, a la cual, le estamos asignando el valor “Hola Mundo !!!”

ejercicio hola mundo

Creamos el controlador

El fichero `myApp.js` queda de la siguiente manera:

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

myApp.controller('MyAppCrtl', ['$scope', function($scope){
    $scope.message = "Hola Mundo!!!!";
}]);
```

ejercicio hola mundo

inyectamos el controlador

Para inyectar el controlador dentro del módulo root utilizaremos dos directivas:

- **ng-app**: Atributo que indica que modulo de angular se inyecta en el root para la vista dada
- **ng-controller**: Atributo que indica que contenedor de tipo controlador se inyecta en el módulo root para la vista dada.

Las condiciones de uso de estas directivas son las siguiente:

- La etiqueta que usa el ng-controller debe de estar contenida dentro de la etiqueta que usa el ng-app
- El controlador referenciado en la etiqueta ng-controller debe de pertenecer al modulo referenciado en la etiqueta ng-app

ejercicio hola mundo

inyectamos el controlador

Modificaciones en el fichero index.html

- **Importamos nuestro fichero myApp.js:** Utilizamos la etiqueta script para añadir el código de nuestro controlador al index.html
- **Inyectamos el módulo:** Utilizamos la directiva ng-app para injectar nuestro modulo dentro de root. El ámbito de ejecución se reduce al html contenido en la etiqueta div
- **Inyectamos el controlador:** Utilizamos la directiva ng-controller para injectar nuestro controlador en el root. El ámbito de ejecución se reduce al html contenido en la etiqueta div

```
<!DOCTYPE html>
<html>
<head>
<script src=".//Lib/angular/angular.min.js"></script>
<script src=".//app/controllers/myApp.js"></script>
<meta charset="UTF-8">
<title>Practicas de angular</title>
</head>
<body>
<div ng-app="MyApp">

  <div ng-controller="MyAppCrtl">
    <h1>{{message}}</h1>
  </div>
</div>
</body>
</html>
```

ejercicio hola mundo

inyectamos el controlador

Modificaciones en el fichero index.html

```
<!DOCTYPE html>
<html>
<head>
<script src=".Lib/angular/angular.min.js"></script>
<script src=".app/controllers/myApp.js"></script>
<meta charset="UTF-8">
<title>Practicas de angular</title>
</head>
<body>
<div ng-app="MyApp">
    <div ng-controller="MyAppCrtl">
        <h1>{{message}}</h1>
    </div>
</div>
</body>
</html>
```



- **Visualizamos la variable message:** Con el doble corchete accedemos al valor de la variable message que declaramos en el scope

ejercicio hola mundo

directiva ng-model

Ahora vamos a modificar nuestro hola mundo para que podamos cambiar el valor de la variable message desde la vista. Para ello, en el fichero index.html añadimos el siguiente input de texto:

```
<div ng-app="MyApp">
  <div ng-controller="MyAppCrtl">
    <h1>{{message}}</h1>
    <input type="text" ng-model="message"/>
  </div>
</div>
```

La directiva ng-model asocia el contenido de un campo a una variable del controlador. Es lo que se conoce como DOM declarativo; donde el desarrollador declara variables que modifican el aspecto de la vista sin necesidad de recorrer las diferentes etiquetas html. Ahora, al desplegar la aplicación y escribir en el cuadro de texto, vemos como cambia el mensaje de la pantalla

ejercicio hola mundo

directiva ng-click

Al scope, no solo se le pueden asignar variables, también se pueden asignar funciones asociadas a eventos que se ejecutan al accionarse. Para ver esto mismo añadamos el siguiente código a nuestro controlador:

```
$scope.showMessage = function (message){  
    alert("Valor del mensaje: " + message);  
};
```

Como vemos, estamos asignando al scope una función que muestra una alerta con el valor de una variable que recibe como parámetro de entrada.

ejercicio hola mundo

directiva ng-click

El siguiente paso es crear un botón que, al pulsarse, ejecute la función que acabamos de crear:

```
<div ng-app="MyApp">
  <div ng-controller="MyAppCrtl">
    <h1>{{message}}</h1>
    <input type="text" ng-model="message"/>
    <button type="submit" ng-click="showMessage(message)">Envia
    mensaje</button>
  </div>
</div>
```

La directiva ng-click se utiliza para ejecutar el código de javascript que contiene cuando se pulsa la etiqueta a la que se asocia; sea esta un botón, una celda de una tabla, un link o cualquier otro elemento html.
Ahora, cada vez que pulsamos el botón “Envia mensaje” aparece una alerta con el título de la pantalla.

ejercicio hola mundo

directiva ng-click

Implementar vosotros mismo el botón envía mensaje



ejercicio creacion de servicios

conceptos básicos

- **Funciones asíncronas:** funciones que son lanzadas en hilos de ejecución diferentes al de la aplicación.
- **Promesas:** Técnica utilizada en javascript para poder sincronizar el código con la ejecución de funciones asíncronas. Básicamente, consiste en encapsular una trozo de código dentro de una función que espera una señal de éxito o fracaso proveniente de una ejecución asíncrona.

ejercicio creacion de servicios

conceptos básicos

El objetivo de este ejercicio es aprender a declarar nuestros propios servicios en angular. Para ello crearemos un servicio que liste una serie de directores de cine que obtendremos de una llamada al back. Además de esto, nuestro servicio deberá proveernos de las funcionalidades necesarias para modificarlos, borrarlos o darlos de alta. Los conocimientos que pondremos en práctica son:

1. La estructura de un servicio
2. Las promesas y su uso
3. El uso de servicios propios de angular
4. Como compaginar los elementos vista, modelo y controlador en angular

ejercicio creacion de servicios

preparación del index.html

En este ejercicio tenemos que construir una interfaz mas sofisticada que en el ejercicio anterior.

Para ello, vamos a incluir las librerías del framework bootstrap.

Tomando como base la solución del ejercicio 1, modificamos el index.html de la siguiente manera:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap-3.2.0-dist/css/bootstrap.min.css"/>
    <link rel="stylesheet" type="text/css" href="./lib/bootstrap/bootstrap-3.2.0-dist/css/bootstrap-theme.min.css"/>
    <script src="./Lib/angular/angular.min.js"></script>
    <script src="./Lib/bootstrap/bootstrap.min.js"></script>
    <script src="./Lib/bootstrap/ui-bootstrap-tpls-0.11.2.min.js"></script>
    <script src="./app/controllers/myApp.js"></script>
    <meta charset="UTF-8">
    <title>Practicas de angular</title>
  </head>
  <body ng-app="MyApp">
    <div>
      <div ng-controller="MyAppCrtl">
        <h1>{{message}}</h1>
        <input type="text" ng-model="message"/>
        <button type="submit" ng-click="showMessage(message)">Envia mensaje</button>
      </div>
    </div>
  </body>
</html>
```

ejercicio creacion de servicios

nuestro primer servicio

De la misma forma que hicimos con el controlador, declaramos un módulo nuevo para almacenar nuestro servicio

```
var restServiceApp = angular.module('DirectorServiceApp',[]);

restServiceApp.factory('DirectorService',['$http','$q',function($http,$q){

function ServiceRest (){
    ...
}

return {'serviceDirector':ServiceRest};
}]);
```

Factory es el método de angular para dar de alta un nuevo servicio. Las inyecciones funcionan de la misma forma que en los controladores

La clase ServiceRest será donde implementaremos la lógica del servicio

Como resultado, el servicio, podría haber devuelto un objeto en vez de la clase en si. Si devolvíramos el objeto, este sería constante para toda la aplicación y cualquier cambio que se hiciera permanecería sobre el objeto hasta que se cerrará la página. Al devolver una clase estamos obligados a instanciar un objeto, lo que significa que, dos accesos diferentes al servicio, obtienen objetos diferentes e independientes

ejercicio creacion de servicios

nuestro primer servicio

```
function ServiceRest (){
  this.url="data.json";

  this.consultaDirectores = function (){
    var defer = $q.defer();
    $http.get(this.url).success(function(data){
      defer.resolve(data);
    }).error (function(error){
      defer.reject(error);
    });
    return defer.promise;
  };

  this.nuevoDirector = function (director){
    var defer = $q.defer();
    $http.post(this.url,director).success(function (data){
      defer.resolve(data);
    }).error(function(error){
      defer.reject(error);
    });
    return defer.promise;
  };
}
```

Si la llamada al servicio falla rechazamos la promesa y devolvemos el error

Este método obtiene el listado de directores

El servicio http es el encargado de realizar las llamadas AJAX al servicio REST

\$q es el servicio de angular encargado de gestionar las promesas. El método defer genera una nueva promesa

Como resultado del método devolvemos el objeto promesa en si

Si la llamada al servicio termina con éxito resolvemos la promesa devolviendo el resultado de la llamada

ejercicio creacion de servicios

Ahora implementar vosotros mismo el servicio DirectorService con la siguiente funcionalidad:

- Un método que permita consultar el listado de directores
- Un método que permita dar de alta un director
- Un método que permita modificar los datos de un director
- Un método que permita borrar un director



ejercicio creacion de servicios

el controlador

```
var myApp = angular.module('MyApp',[ 'DirectorServiceApp']);  
  
myApp.controller('MyAppCrtl',['$scope','DirectorService',function($scope,DirectorService){  
    var directorService = new DirectorService.serviceDirector();  
  
    function getDirectores (){  
        directorService.consultaDirectores().then(function (data){  
            $scope.directores = data;  
        },function (error){  
            alert(JSON.stringify(error));  
        });  
    };  
  
    getDirectores();  
  
});  
...  
}]);
```

El primer cambio que hacemos en el controlador es injectar el módulo que contiene el servicio

Después inyectamos el propio servicio

Y, por supuesto, instanciamos un objeto del propio servicio

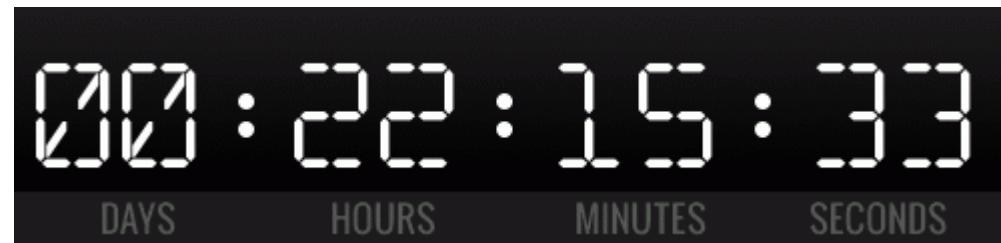
Then es el método que espera el resultado de las promesas. Recibe dos funciones como parámetros

La primera función se ejecuta si la promesa se resuelve con éxito

ejercicio creacion de servicios

Ahora añadir las modificación al controlador para que la vista pueda implementar las funcionalidades de consulta, alta, modificación y borrado de un director. Para ello debéis:

- Declarar en el scope una variable donde se almacenará el objeto de un director seleccionado
- Declarar en el scope una función save que guarde los cambios realizados sobre un director. Esta función debe de seguir la siguiente lógica:
- Declarar en el scope una función remove que borre un director.



ejercicio creacion de servicios

```
<div class="col-md-12">
<table class="table table-bordered">
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Apellidos</th>
    </tr>
  </thead>
  <tbody>
    <tr ng-repeat="director in directores">
      <td>{{director.nombre}}</td>
      <td>{{director.Apellidos}}</td>
      <td><button type="submit" ng-click="seleccionar(director)">seleccionar</button></td>
    </tr>
  </tbody>
</table>
</div>
<div class="col-md-6">
  <br/><br/>
  <input type="text" ng-model="seleccionado.nombre"/>
</div>
<div class="col-md-6">
  <br/><br/>
  <input type="text" ng-model="seleccionado.Apellidos"/>
</div>
<div class="col-md-6">
  <br/><br/>
  <button type="submit" ng-click="save(seleccionado)">Guardar</button>
</div>
<div class="col-md-6">
  <br/><br/>
  <button type="submit" ng-click="remove(seleccionado)">Borrar</button>
</div>
```

Esta es la tabla donde pintamos el listado de directores

Este es el bucle que pinta cada uno de los directores en la tabla

Con este botón seleccionamos un director

Con este otro actualizamos los cambios

Y con este otro borramos al director seleccionado

ejercicio creacion de servicios

Como siguiente paso del ejercicio modificaréis el fichero index.html para que pinte una tabla con el listado de directores además de, un formulario para:

- Dar de alta un nuevo director
- Modificar los datos de un director previamente seleccionado
- Borrar los datos de un director previamente seleccionado



ejercicio creacion de filtros

El objetivo de este ejercicio es aprender que son los filtros en angular y como funcionan. Para ello, crearemos un filtro que, a partir de una cadena de texto introducida en un input, nos elimine de la tabla todos aquellos directores cuyo nombre y apellido no contiene dicha cadena. Los conocimientos que podremos en practica son:

- Creación de un filtro en angular
- Utilización de los filtros en las vistas

ejercicio creacion de filtros

Como ya hicimos otras veces, declaramos nuestro filtro dentro de un nuevo módulo

```
var directorFilterApp = angular.module('DirectorFilter',[]);

directorFilterApp.filter('DirectorFilter',function(){
    return function (directores,cadena){
        var result [];
        if ((directores) && (cadena)){
            for (var i=0;i<directores.length;i++){
                if ((directores[i].nombre.indexOf(cadena)>=0) || (directores[i].Apellidos.indexOf(cadena)>=0)){
                    result.push(directores[i]);
                }
            }
        }
        return result;
    } else {
        return directores;
    }
});
```

Filter es el método de angular para declarar nuevos filtros. Las inyecciones funcionan igual que en los servicios y controladores

Lo que un filtro devuelve es el resultado de la ejecución de una función que recibe, como mínimo, un parámetro a transformar. Dicho parámetro tiene que ser el primero.

ejercicio creacion de filtros

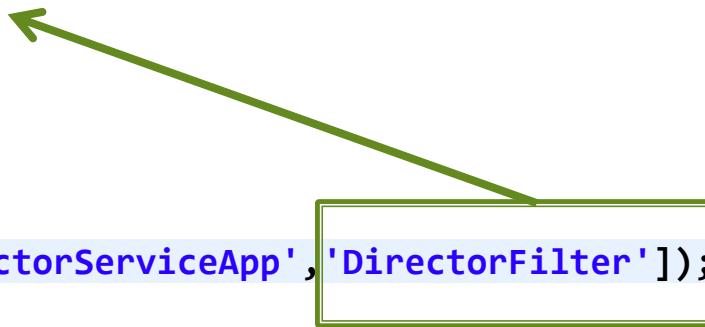
Ahora implementar vosotros mismos un filtro que reciba dos parámetros. Un listado de directores y una cadena. Como resultado, la función devolverá el listado de directores cuyo nombre o apellidos contengan la cadena de caracteres. Tienes que tener en cuenta que el filtro se va a ejecutar siempre que se cargue la página, por lo tanto, puede ser que el listado de directores y la cadena te vengan vacíos, nulos o indefinidos. Contempla también esas posibilidades.

ejercicio creacion de filtros

el controlador

Un filtro puede usarse desde cualquier contendor de angular, como un controlador o un servicio. Sin embargo esto no suele ser común. La ventaja de los filtros es que pueden ser ejecutados desde las vistas sin necesidad de que otro elemento intervenga. Solamente se necesita incluir el módulo del filtro en el controlador que pinta la vista.

```
var myApp = angular.module('MyApp',[ 'DirectorServiceApp' , 'DirectorFilter']);
```



ejercicio creacion de filtros

Index.html

```
<div class="col-md-6">
    <label for="filtro">Filtrado de directores: </label>
</div>
<div class="col-md-6">
    <input id="filtro" type="text" ng-model="filtrado"/>
</div>
<div class="col-md-12">
    <br/>
    <table class="table table-bordered">
        <thead>
            <tr>
                <th>Nombre</th>
                <th>Apellidos</th>
            </tr>
        </thead>
        <tbody>
            <tr ng-repeat="director in directores|DirectorFilter:filtrado">
...
...
```

Creamos un campo de texto para introducir la cadena por la que filtrar

El operador que llama a los filtros es el “|”. A la izquierda del pipe siempre se coloca el primer parámetro de la función del filtrado mientras que, a la derecha, va el nombre del filtro. En caso de necesitar más parámetros se irán colocando a la derecha del nombre del filtro separados por “:”

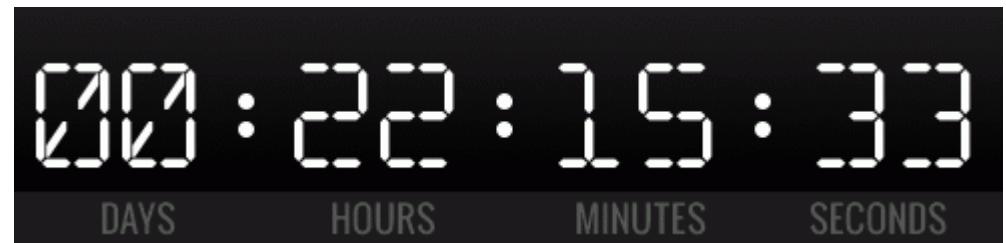
ejercicio creacion de filtros

el controlador

Modificar el controlador y el fichero index.html para que se pueda filtrar los directores de la tabla por una cadena de texto introducida desde un campo input.

Ejercicio adicional:

Crea un filtro para que, activando o desactivando un botón de la interfaz, todos los nombres y apellidos de los directores de cine aparezcan en mayúsculas o minúsculas



ejercicio creacion de directivas

el controlador

- **Plantilla:** Pedazo de código en html usado por un controlador o una directiva para componer un interfaz web

El objetivo de este ejercicio es aprender que son las directivas y para que sirven. Para ello, crearemos una directiva que encapsule toda la funcionalidad de la tabla. Los conocimientos que podremos poner en práctica son:

- Creación de plantillas HTML
- Creación de directivas básicas
- Compartimentación del contexto del controlador

ejercicio creacion de directivas

La plantilla

```
<div class="row">
  <div class="col-md-4">
    <label for="filtro">Filtrado de directores: </Label>
  </div>
  <div class="col-md-4">
    <input id="filtro" type="text" ng-model="tablaInfo.filtrado"/>
  </div>
  <div class="col-md-4">
    <button type="button" ng-click="tablaInfo.changeCaps()" class="{{tablaInfo.styleButton}}>Mayusculas</button>
  </div>
  <div class="col-md-12">
    <br/>
    <table class="table table-bordered">
      <thead>
        <tr>
          <th ng-repeat="head in tablaInfo.headers">{{head}}</th>
        </tr>
      </thead>
      <tbody>
        <tr ng-repeat="director in tablaInfo.data/DirectorFilter:tablaInfo.filtrado">
          <td ng-repeat="head in tablaInfo.headers">
            {{director[head]}|ChangeCaps:tablaInfo.caps}}
          </td>
          <td><button type="submit" ng-click="tablaInfo.seleccionar(director)">seleccionar</button></td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="col-md-12">
    <div style="text-align:center">
      <button type="button" class="btn btn-primary" ng-click="tablaInfo.ant()>Anterior</button>&ampnbsp
      <button type="button" class="btn btn-primary" ng-click="tablaInfo.sig()>Siguiente</button>&ampnbsp&ampnbsp
      <select ng-model="tablaInfo.size">
        <option value="5">5</option>
        <option value="10">10</option>
        <option value="15">15</option>
      </select>
      <button type="button" class="btn btn-default" ng-click="tablaInfo.actualizar(tablaInfo.size)">Actualizar</button>
    </div>
  </div>
</div>
```

Todo este código HTML engloba la lógica que hemos ido añadiendo a la tabla en los diferentes ejercicios. Desde el filtrado hasta la paginación de los resultados

El contexto del controlador de la directiva lo vamos a compartimentar. De esta manera, si utilizamos mas de una tabla en la misma vista, evitaremos conflictos por usar las mismas variables en el mismo contexto. El compartimento en el que la directiva almacena sus variables/ eventos se llamará tablaInfo

ejercicio creacion de directivas

nuestra primera directiva

Esto indica que nuestra directiva se identificará por un atributo de HTML en vez de por una etiqueta. La razón de esto es para evitar incompatibilidades con IE8

```
var tableDirectiveApp = angular.module("TableDirectiveApp",[]);  
  
tableDirectiveApp.directive('myTable',function(){  
    return{  
        restrict:'A',  
        templateUrl: './app/templates/templateTable.html',  
        scope:{  
            tablaInfo:'=tabInfo'  
        }  
    };  
});
```

El objeto que definirá el comportamiento del contexto en el scope de la directiva se le pasará a esta mediante el atributo tab-info

Como de costumbre, declaramos un nuevo módulo para la directiva

El nombre del atributo que lanza nuestra directiva se define por su propia referencia. Como en HTML no se distingue entre mayúsculas y minúsculas, la nomenclatura tradicional de javascript no sirve a la hora de declarar directivas. Es por ello que, angular, traducirá 'myTable' como el atributo 'my-table'

ejercicio creacion de directivas

el controlador

```
var myApp = angular.module('MyApp',['DirectorServiceApp','DirectorFilter','TableDirectiveApp']);

myApp.controller('MyAppCtrl',['$scope','DirectorService',function($scope,DirectorService){
    var directorService = new DirectorService.serviceDirector();

    $scope.tabla1={
        headers:[],
        data:[],
        seleccionar:function(director){
            $scope.seleccionado=director;
        },
        sig:function (){
            directorService.siguiente();
            getDirectores();
        },
        ant:function (){
            directorService.anterior();
            getDirectores();
        },
        size:5,
        changeCaps:function (){
            if ($scope.tabla1.caps == "mins"){
                $scope.tabla1.caps="mays";
                $scope.tabla1.styleButton="btn btn-primary active";
            } else {
                $scope.tabla1.caps="mins";
                $scope.tabla1.styleButton="btn btn-default";
            }
        },
        filtrado:"",
        styleButton:"btn btn-default",
        caps:"mins",
        actualizar: function(size){
            directorService.cambiaTamanyo(size);
            getDirectores();
        }
    };
    ...
}]);
```

TableDirectiveApp'



Lo primero: Inyectar el módulo de la directiva en el controlador

Todas las funciones y variables que habíamos definido para la tabla (funciones de paginación, seleccionar director, cambiar el tamaño de la página, etc...) ahora, deben de definirse dentro de un objeto del scope que será el compartimento de la directiva



ejercicio creacion de directivas

el controlador

```
<div ng-controller="MyAppCrtl">
  <div my-table="" tab-info="tabla1"></div>
  <div class="row">
    <div class="col-md-6">
      <br/><br/>
      <input type="text" ng-model="seleccionado.nombre"/>
    </div>
    <div class="col-md-6">
      <br/><br/>
      <input type="text" ng-model="seleccionado.Apellidos"/>
    </div>
    <div class="col-md-6">
      <br/><br/>
      <button type="submit" ng-click="save(seleccionado)">Guardar</button>
    </div>
    <div class="col-md-6">
      <br/><br/>
      <button type="submit" ng-click="remove(seleccionado)">Borrar</button>
    </div>
  </div>
</div>
```

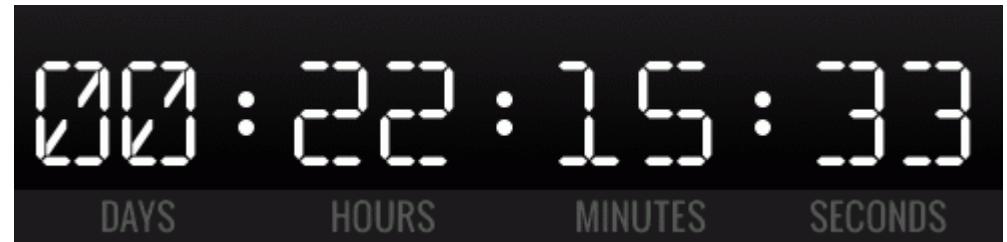
Aquí es donde definimos que contiene el
compartimiento de la directiva

Todo el código html de la tabla
es sustituido por esta línea

ejercicio creacion de directivas

el controlador

Ahora, siguiendo el ejemplo visto en las anteriores trasparencias, cread vosotros mismos una directiva que abarque toda la funcionalidad de la tabla.



ejercicio routing

conceptos básicos

- **Router:** aplicación de javascript encargada de interceptar las peticiones http que lanza el navegador y ejecutar la acción adecuada a dicha petición
- **Estado:** Conjunto de vista mas funcionalidad que se da en el navegador a razón de un evento
- **Estado abstracto:** Estado encargado de pintar un contexto común a un grupo de estados hijos. Dentro del árbol de estados de una aplicación no puede ser hoja
- **Estado vista:** Estado que aglutina un conjunto de dos o mas vistas. Solo puede ser estado hoja
- **Estado padre:** Estado que comparte su contexto con una serie de estados llamados hijos. Los estados hijos pueden consultar las variables del estado padre pero no pueden escribir las

ejercicio routing

conceptos básicos

Hasta ahora, nuestra aplicación, no ha dejado de ser una aplicación de escritorio, es decir, en ningún momento hemos navegado a otra ventana. Siempre hemos permanecido en la misma url. El objetivo de esta lección es aprender como se pueden implementar la navegación entre ventanas a través de angular. Para ello utilizaremos el modulo ui-router que implementa la navegación como una maquina de estados, entre los cuales, se pueden establecer relaciones de parentesco similares a las de la herencia de objetos. Los conocimientos que podremos en práctica son:

- Construcción y configuración de un router
- Creación de una maquina de estados
- Aprender el uso de los distintos tipos de estados

ejercicio routing

el router

```
var router = angular.module('Router', [ 'ngRoute' ])  
  .config(function($routeProvider){  
    $routeProvider  
      .when('/', {  
        templateUrl: "./app/templates/barraMenu.html",  
        abstract: true,  
        controller: "menuCrtl"  
      })  
      .when('/consultaDirectores', {  
        templateUrl: "./app/templates/plantillaDirectores.html",  
        controller: "MyAppCrtl"  
      });  
});
```

El módulo del ngRoute que nos permite implementar la navegación entre páginas

Y los restantes módulos con los controladores de nuestra aplicación

Y una template html que pintar

Además, pueden tener un controlador que es ejecutado cada vez que se accede al estado

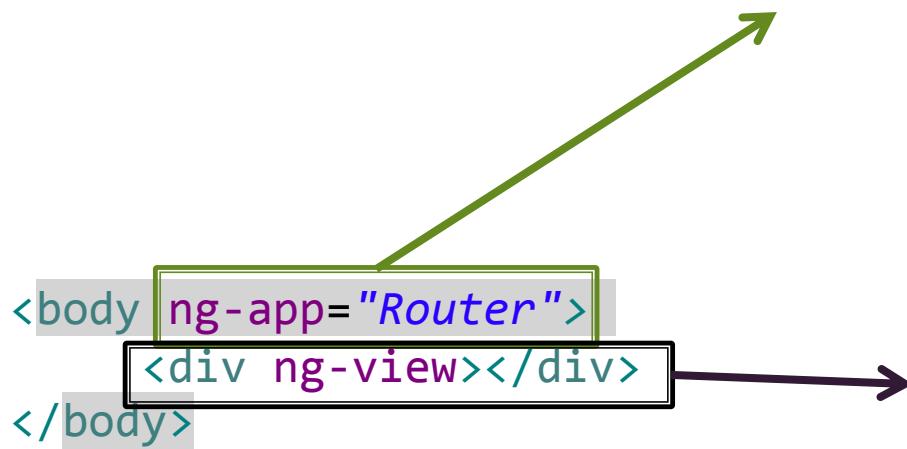
Estos son los nombres de las URL

ejercicio routing

el router

Ahora, nuestro fichero index.html, forma parte del router.

El módulo que inyectamos en el contexto root es el módulo del router y, será este, el encargado de injectar los diferentes módulos de los diferentes controladores según se vayan necesitando



Todo el código de HTML que tenia el index se ha sustituido por un div con la etiqueta `ng-view`. Será dentro de este div donde el router pintara las diferentes plantillas asociadas a los diferentes estados según se vaya accediendo a ellos

ejercicio routing

estado abstracto

El router no considera un estado abstracto como un estado navegable. Es decir, no es posible acceder solamente a él. Su misión es formar un marco donde otros estados puedan ejecutarse. En consecuencia, esta es la plantilla que tiene asociada:

```
<div class="navbar navbar-default" role="navigation">
  <div class="navbar-header">
    <a class="navbar-brand" href="#">{{barra.name}}</a>
  </div>

  <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
    <div ng-repeat="menu in barra.menus">
      <ul class="nav navbar-nav">
        <li id="{{menu.id}}" class="dropdown">
          <a class="dropdown-toggle" data-toggle="dropdown">{{menu.name}} <b class="caret"></b></a>
          <ul class="dropdown-menu">
            <li ng-repeat="option in menu.options"><a href="{{option.href}}>{{option.name}}</a></li>
          </ul>
        </li>
      </ul>
    </div>
  </div><!-- /.navbar-collapse -->
</div>
```

Esto es una barra de menú donde se plasman las diferentes opciones de navegación de la web

ejercicio routing

estado hijo

```
var menuApp = angular.module("MenuApp",[]);

menuApp.controller('menuCrtl',['$scope','$http',function($scope,$http){

$scope.barra={
  "name":"Directores de cine",
  "menus":[{
    "name":"Directores",
    "options":[{
      "name":"Consulta directores",
      "href": "#/consultaDirectores"
    }]
  }]
};

}]);
```

En el controlador del menu añadimos el nombre de la nueva ruta a la propiedad href para que sea posible navegar a él desde la barra de menús