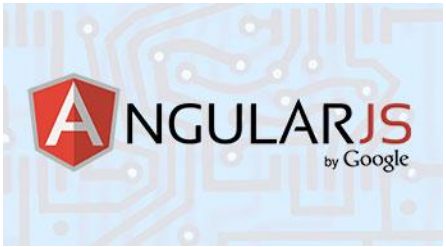

BEEVA

BEE PART OF THE CHANGE

Avenida de Burgos 16 D, 28036 Madrid
hablemos@beeva.com
www.beeva.com

# DOCUMENTACIÓN

# NIVEL BÁSICO

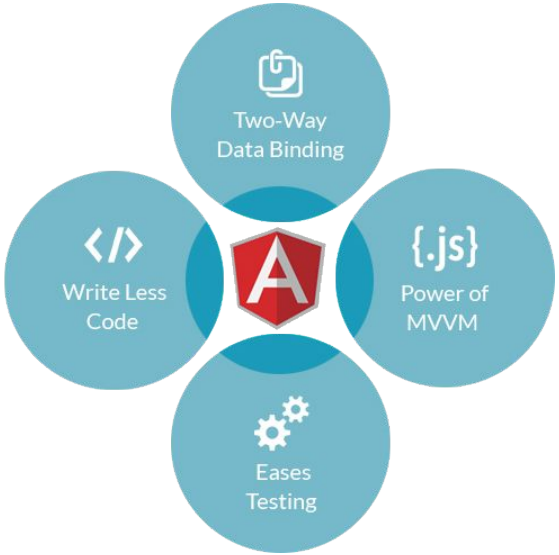



INTRODUCCIÓN
DATA BINDING
CONTROLADORES
SCOPE
INYECCIÓN DE DEPENDENCIA
EXPRESIONES
DIRECTIVAS NATIVAS
CONTROL DE ERRORES
BUENAS PRÁCTICAS

# INTRODUCCIÓN

# Introducción

- Angular es un framework Javascript del lado cliente (client - side) que proporciona una solución global para crear aplicaciones web dinámicas
- Enseña nueva sintaxis al navegador, para darle más funcionalidad
- Utiliza la forma SPA (Single page application)
- Patrones MVC, MVVM, MVW
- Compatible con cualquier tecnología de servidor
- Facilmente testeable (unitarios, extremo a extremo)



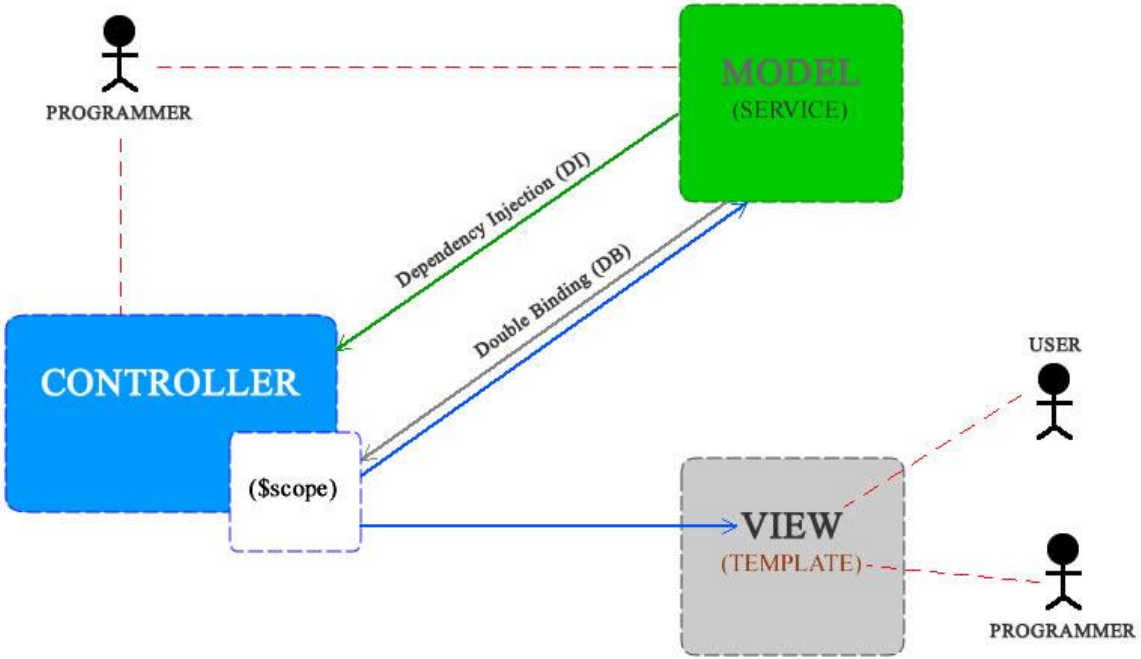
# Introducción

## Conceptos

- **Template (Plantilla)**: HTML con valores añadidos
- **Directives (Directivas)**: HTML extendido con atributos y elementos personalizados
- **Model (Modelo)**: los datos que se mostrarán en la vista y con los cuales el usuario interactúa
- **Scope**: contexto donde se almacena el modelo para que controladores, directivas y expresiones puedan acceder a él
- **Expressions (Expresiones)**: acceso a variables y funciones desde el scope, enlazando datos a la vista
- **Filters (Filtros)**: formatea el valor de una expresión para mostrarla al usuario
- **View (Vista)**: lo que el usuario ve (el DOM)
- **Data binding**: sincronización de los datos entre el modelo y la vista
- **Controller (Controlador)**: la lógica de negocio que hay detrás de la vista, conectándola con el modelo
- **Inyección de dependencias**: crea y deja disponibles objetos y funciones
- **Module (Módulo)**: contenedor para la diferentes partes de la aplicación incluyendo controladores, servicios, filtros, directivas, constantes, factorías...
- **Service (Servicio)**: lógica de negocio reusable e independiente de las vistas

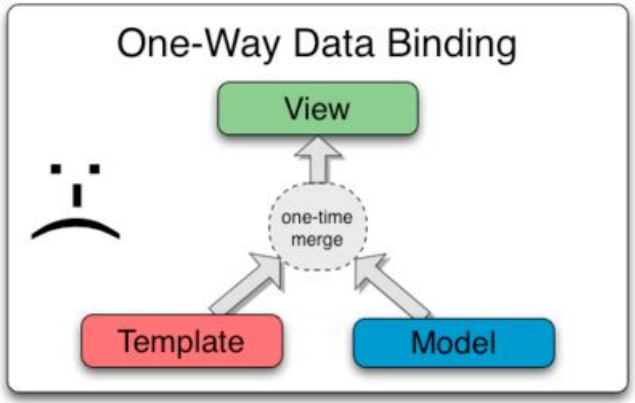
# Introducción

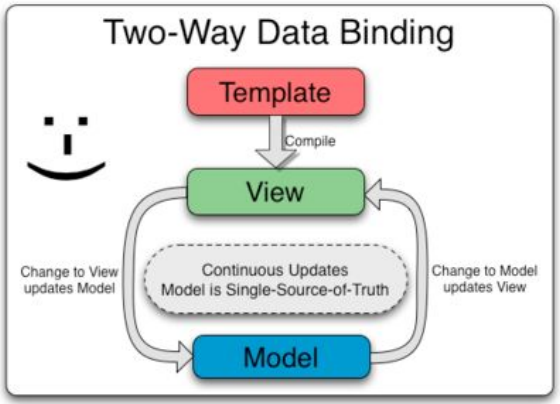
## Conceptos

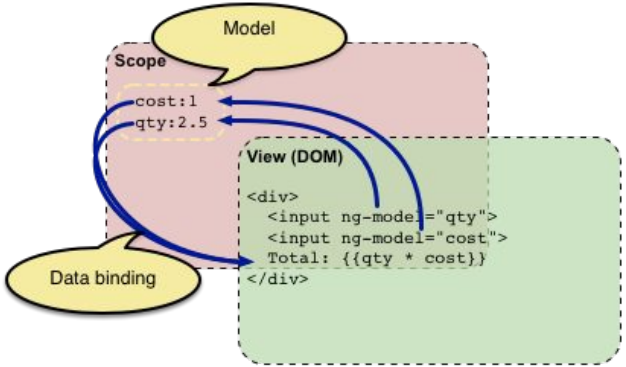


# DATA BINDING

# Data Binding



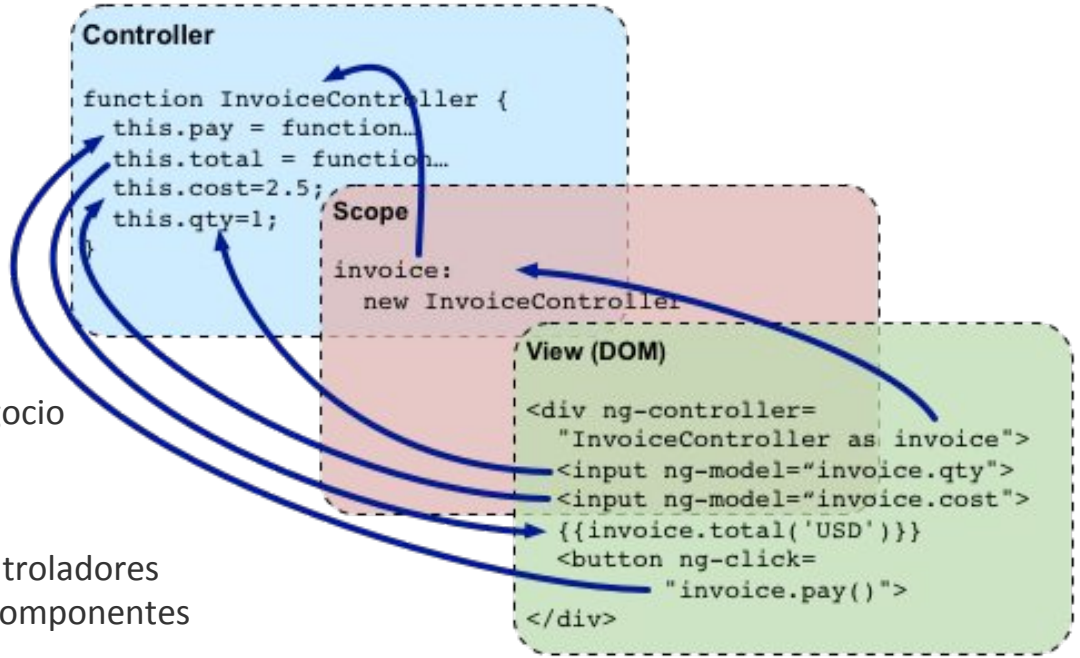



# CONTROLADORES

# Controladores

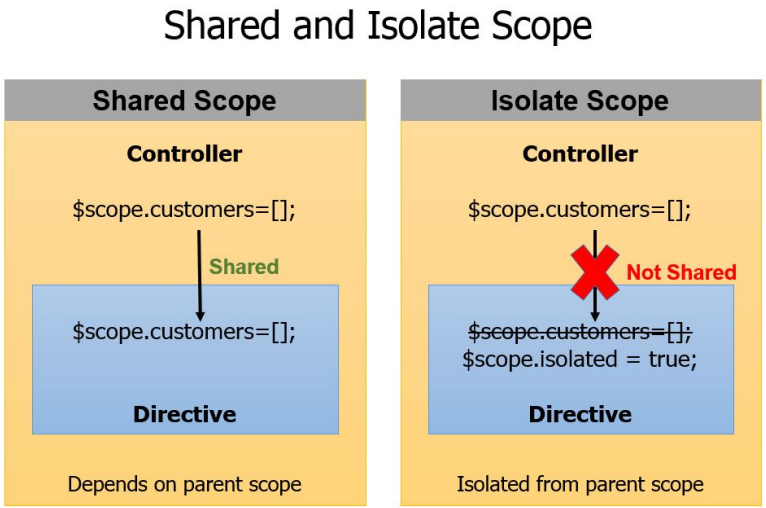
- En Angular se usan para aumentar el scope
- Se adjunta en el DOM con *ng-controller*
- Usaremos controladores para:
  - Establecer el estado inicial del scope
  - Añadir comportamiento al scope
- No los usaremos para:
  - Manipular el DOM. Solo lógica de negocio
  - Formatear  inputs
  - Filtrar las salidas
  - Compartir código o estados entre controladores
  - Gestionar los ciclos de vida de otros componentes



# SCOPE

# Scope

El scope es un contexto donde se almacena el modelo para que controladores, directivas y expresiones puedan acceder a él.

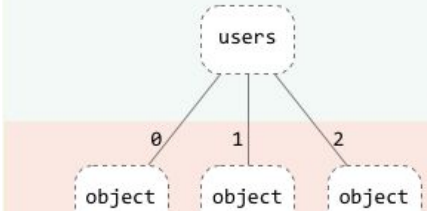



- Hay dos tipos: *child scope* y *isolate scope*
- Proveen un contexto en el cual las expresiones pueden ser evaluadas
- Los scopes nos proveen de APIs (**$watch, $watchCollection, $watchGroup**) para observar las mutaciones del modelo
- Los scopes nos proveen de APIs (**$apply**) para propagar cualquier cambio del modelo

# Scope

## Observadores

```
$scope.users = [
  {name: "Mary", points: 310},
  {name: "June", points: 290},
  {name: "Bob", points: 300}
];
```




**By Reference**

```
$scope.$watch("users", …);
✔ $scope.users = newUsers;
✘ $scope.users.push(newUser);
✘ $scope.users[0].points = 320;
```

**By Collection Items**

```
$scope.$watchCollection("users", …);
✔ $scope.users = newUsers;
✔ $scope.users.push(newUser);
✘ $scope.users[0].points = 320;
```

**By Value**

```
$scope.$watch("users", …, true);
✔ $scope.users = newUsers;
✔ $scope.users.push(newUser);
✔ $scope.users[0].points = 320;
```

# Scope
## Alias

- Modalidad alternativa de enviar datos al Scope
- controller...as

```
<div ng-controller="ejemploCtrl as ejemplo">
```

- A parte de poder utilizar el objeto $scope, se puede utilizar **this** dentro de la función

```
.controller('ejemploCtrl', function($rootScope){
    var model = this;
    model.data = "12";
 });
```

- Usando el alias se puede acceder al dato de la siguiente manera desde la plantilla

```
{{ ejemplo.data }}
```

# Scope

## Jerarquías

- Cada aplicación Angular tiene un *$rootScope,* pero puede tener muchos *child scope*
- Angular inserta automáticamente una clase *ng-scope* en los elementos donde se adjuntan scopes
- El $rootScope se aloja en el DOM con la directiva *ng-app*
- $parent para acceder a controladores superiores o raíz

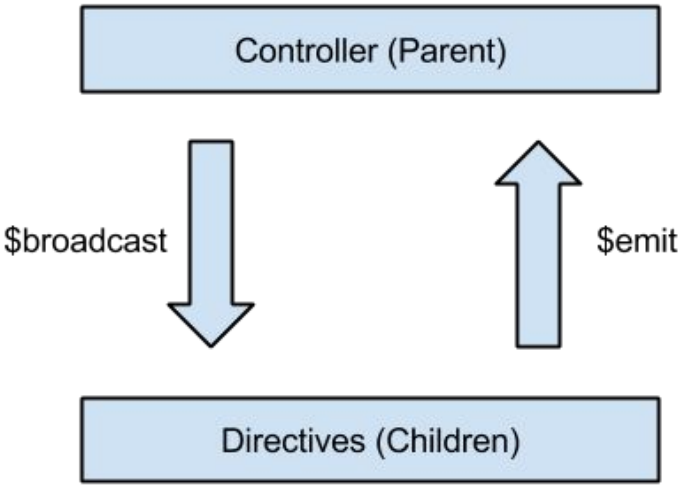

* A modo debug, poniendo *angular.element($0).scope()* en la consola obtendremos el scope asociado a la etiqueta

* También se puede utilizar la herramienta Angular Batarang para Chrome

# Scope

## Eventos

Los scopes pueden propagar eventos de manera similar a los eventos del DOM




- **$broadcast**: transmite un evento hacia todos los *child scopes* a partir del scope desde donde se lanza
- **$emit**: emite un evento hacia los *parent scopes* a partir del scope desde donde se lanza
- **$on**: escucha cualquier evento que pueda ser lanzado y permite adjuntar una función (*listener*)

# Scope

## Eventos (ejemplo)

### PLANTILLA HTML

```
<body>
        <!--controlador padre que contiene homeCtrl y profileCtrl-->
        <div ng-controller="appCtrl">
                {{ count }}
                <button ng-click="updateCounter1()">Incrementar hijos desde appCtrl</button>

                <!--controlador hijo de appCtrl-->
                <div ng-controller="homeCtrl">
                        {{ count }}
                        <button ng-click="updateCounter2()">Incrementar app desde home</button>
                </div>

                <!--controlador hijo de appCtrl-->
                <div ng-controller="profileCtrl">
                        {{ count }}
                        <button ng-click="updateCounter3()">Incrementar app desde profile</button>
                </div>

        </div>
</body>
```

# Scope

## Eventos (ejemplo)

### CONTROLLERS (I)

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

.controller('appCtrl', ['$scope', function ($scope)
{
        //inicializamos count
        $scope.count = 0;

        //al pulsar el botón llamamos al evento to_childrens para actualizar todos los hijos
        $scope.updateCounter1 = function()
        {
                $scope.$broadcast('to_childrens', 1);
        }

        $scope.$on('to_parent', function(event, data)
        {
        $scope.count += data;
    });
}])
```

# Scope

## Eventos (ejemplo)

### CONTROLLERS (II)

```
.controller('homeCtrl', ['$scope', function ($scope)
{
        $scope.count = 0;
        $scope.$on("to_childrens", function(event, data)
        {
                $scope.count += data;
        })

        $scope.updateCounter2 = function()
        {
                $scope.$emit('to_parent', 1);
        }
}])
```

# Scope

## Eventos (ejemplo)

### CONTROLLERS (III)

```
.controller('profileCtrl', ['$scope', function ($scope)
{
        $scope.count = 0;
        $scope.$on("to_childrens", function(event, data)
        {
                $scope.count += data;
        })

        $scope.updateCounter3 = function()
        {
                $scope.$emit('to_parent', 1);
        }
}])
```

# INYECCIÓN DE DEPENDENCIAS

# Inyección de dependencias (DI)

- Patrón de diseño orientado a objetos
- Especifica los objetos que serán suministrados a una determinada clase
- Angular hace un uso intensivo de este patrón

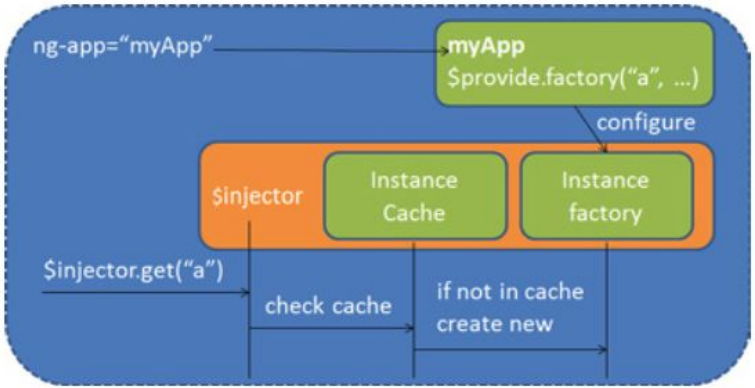



Imagen: https://dzone.com/refcardz/angularjs-essentials

# Inyección de dependencias (DI)

## Anotaciones

### Inline Array Annotation:

```
someModule.controller('MyCtrl', ['$scope', 'greeter', function($scope, greeter) {}]);
```

### Implicit Annotation:

```
someModule.controller('MyController', function($scope, greeter) {});
```

*Cuidado! si tienes pensado minificar el código, los nombres de los servicios se renombrarán y perderás la referencia, haciendo que la aplicación deje de funcionar.

### $inject Property Annotation:

```
var MyController = function($scope, greeter) {}
MyController.$inject = ['$scope', 'greeter'];
someModule.controller('MyCtrl', MyController);
```

# EXPRESIONES

# Expresiones

- Angular une datos al HTML usando para ello expresiones
- Se escriben directamente en la plantilla
- Son fragmentos de código Javascript que se colocan generalmente entre llaves

  **{{expression}}**

- También se puede utilizar la directiva ng-bind, cuya funcionalidad es la misma

  **ng-bind=”expression”**

- Pueden contener cadenas, números, operadores, objetos y variables
- Aceptan filtros pero no condicionales, bucles, excepciones, expresiones regulares, declaración de funciones, creación de objetos, comas, bitwise operators (&, |, ^, ~, <<, >>, >>>), void
- Binding único (sólo una vez) y normal (siempre)

# Expresiones

## Ejemplos

**Cadenas**: `<span>{{“Lucía tiene el código ” + identificador}}</span>`

**Números**: `<span>1+2={{1+2}}</span>`

**Objetos**: `<p>El nombre es {{persona.nombre}}</p>`

**Arrays**: `<p>La ciudad elegida es {{ciudad[2]}}</p>`

**Tipos de binding**:

```
<p id="one-time-binding-example">One time binding: {{::name}}</p>
<p id="normal-binding-example">Normal binding: {{name}}</p>
```

# DIRECTIVAS

# Directivas

- La forma que la que extendemos nuestro HTML
- Permiten agregar desde pequeños trozos de código hasta la funcionalidad más completa


Imagen: http://prendedor.es/creacion-directivas-personalizadas-angularjs/

# Directivas

## Normalización

```
<div ng-controller="Controller">
  <span ng-bind="name"></span>
  <span ng:bind="name"></span>
  <span ng_bind="name"></span>
  <span data-ng-bind="name"></span>
  <span x-ng-bind="name"></span>
</div>
```

El proceso de normalización es el siguiente:

1. Se elimina **x-** y **data-** del elemento/atributo
2. Se convierte **:**, **-** o **_** a **camelCase (lowerCamelCase)**

Es preferible usar la forma ***dash-delimited***, pero si quieres usar una herramienta de validación HTML puedes usar la forma ***data-dash-delimited***. El resto de formas son aceptadas por razones de compatibilidad, pero se desaconseja su uso.

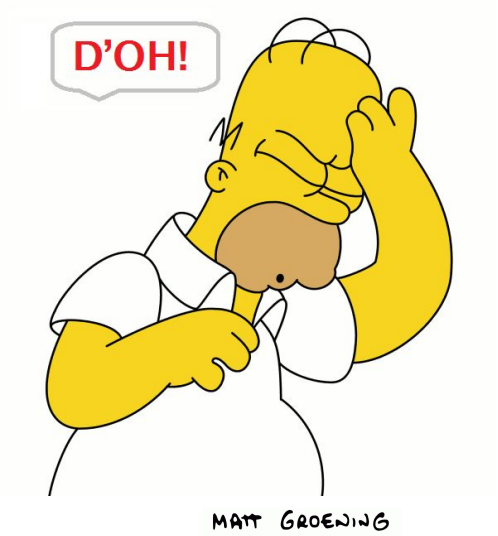
# Directivas

## Nativas

Para una lista completa de directivas nativas de Angular, pincha aquí

- **ngApp**: auto-arranca la aplicación
- **ngDisabled**: establece el atributo *disabled*
- **ngChecked**: establece el atributo *checked*
- **ngReadonly**: establece el atributo *readonly*
- **ngSelected**: establece el atributo *selected*
- **ngForm**: alias de *form*. Permite anidamiento
- **ngValue**: para vincular el valor de *option* o *radios*
- **ngBind**: reemplaza el contenido del elemento
- **ngChange**: evalúa la expr cuando cambia el input
- **ngClass**: establece clases css
- **ngController**: vincula un controlador a la vista
- **ngClick**: se ejecuta cuando el usuario hace click
- **ngIf**: borra o recrea una porción del DOM
- **ngInclude**: extrae, compila e incluye un HTML externo
- **ngInit**: evalúa expresiones en el scope actual
- **ngModel**: vincula el modelo con un *input*, *select*, etc
- **ngOptions**: genera elementos *option* para el *select*
- **ngRepeat**: instancia un template una vez por item
- **ngShow**: muestra u oculta el elemento HTML
- **ngHide**: muestra u oculta el elemento HTML

# CONTROL DE ERRORES



# CONTROL DE ERRORES I (formularios)

**Angular dispone de un sistema muy ágil para el manejo de errores**

**A través de directivas nativas, podremos controlar diferentes casuísticas según el tipo de campo a tratar**

## ESTADOS

- $valid (true / false)
- $invalid (true / false)
- $dirty (true / false)
- $pristine (true / false)

```
<ng-form name='form'>
        <div class=''>
                <label for='nombre'>Nombre</label>
                <input type='text' class='' name='nombre' id='nombre' ng-model='model' required>

                <span ng-show='!form.$pristine && form.nombre.$error.required'>
                        El nombre de usuario es obligatorio
                </span>
        </div>
</ng-form>
```

# CONTROL DE ERRORES II (formularios)

## TIPOS DE ERRORES

- required: campo obligatorio

```
<input type="text" id="field" name="field" ng-model="model.value" required />
```

$error.required

```
<span ng-show="form.field.$error.required">Campo obligatorio</span>
```

- type = email: campo de tipo correo

```
<input type="email" id="field" name="field" ng-model="model.value" required />
```

$error.email

```
<span ng-show="form.field.$error.email">Debe ser una dirección de email válida</span>
```

# CONTROL DE ERRORES III (formularios)

## TIPOS DE ERRORES

- type = text, pattern = '': expresión regular

```
<input type="text" id="field" name="field" ng-model="model.value" ng-pattern="/^[A-Za-zñÑáéíóúÁÉÍÓÚçÇ ]+$/" />
```

$error.pattern

```
<span ng-show="form.field.$error.pattern">Debe escribir un valor válido</span>
```

- type = number, min = '', max = '': mínimo y máximo para un valor numérico

```
<input type="number" id="field" name="field" ng-model="model.value" min="1" max="1000000"  />
```

$error.min, $error.max

```
<span ng-show="form.field.$error.min">Número mínimo no alcanzado</span>
<span ng-show="form.field.$error.max">Número máximo superado</span>
```

# BUENAS PRÁCTICAS

# BUENAS PRÁCTICAS I

**Veamos algunos consejos a la hora de empezar con un proyecto en Angular, qué cosas debemos tener en cuenta para que el desarrollo vaya fluido, y podamos anticiparnos a los diferentes problemas que puedan ir surgiendo:**

- Olvidarse de Jquery (jqLite)
- Evitar el uso del $rootScope en la medida de lo posible
- Utilizar POO (Objetos para los modelos de datos)
- Cuidado con los nombres que utilizamos en cada componente!!!!!!
- Distinguir entre funciones locales y las que definiremos en el scope (también para variables)

# BUENAS PRÁCTICAS II

- Crear servicios para encapsular las llamadas http
- Familiarizarse con la API de Angular (docs.angularjs.org/api)
- Evitar manipular el DOM desde los controladores (directivas)
- En formularios, no usar ng-click y ng-submit a la vez
- Fundamental los test de pruebas unitarios y de integración

# Todo esto está muy bien, pero...

# ¿Cómo empezamos?

## Controlador

```
'use strict';
var myApp = angular.module('spicyApp1', []);
myApp.controller('SpicyController', ['$scope', function($scope) {
  $scope.spice = 'very';
  $scope.chiliSpicy = function() {
    $scope.spice = 'chili';
  };
  $scope.jalapenoSpicy = function() {
    $scope.spice = 'jalapeño';
  };
}]);
```

## Template

```
<html ng-app=”spicyApp1”>
  <body ng-controller="MyController">
    <div ng-controller="SpicyController">
      <button ng-click="chiliSpicy()">Chili</button>
      <button ng-click="jalapenoSpicy()">Jalapeño</button>
      <p>The food is {{spice}} spicy!</p>
    </div>
    <script src="angular.js">
  </body>
</html>
```

# BIBLIOGRAFÍA


- Página principal de Angular: https://angularjs.org/

- Información general (en español):
  http://cursoangularjs.es/doku.php
  http://www.desarrolloweb.com/manuales/manual-angularjs.html
  https://uno-de-piera.com/eventos-en-angularjs/

- Información general (en inglés):
  http://www.w3schools.com/angular/
  http://jimhoskins.com/2012/12/17/angularjs-and-apply.html

- Blog de Beeva:
  https://www.beeva.com/beeva-view/tecnologia/buenas-practicas-de-angular-en-nimble/
  https://www.beeva.com/beeva-view/tecnologia/angular-en-nimble-trabajando-con-clases-objetos-e-instancias/


hablemos@beeva.com
www.beeva.com

# Fernando Castro García

Software Analyst in BEEVA

fernando.castro@beeva.com


hablemos@beeva.com

www.beeva.com