



# Angular JS

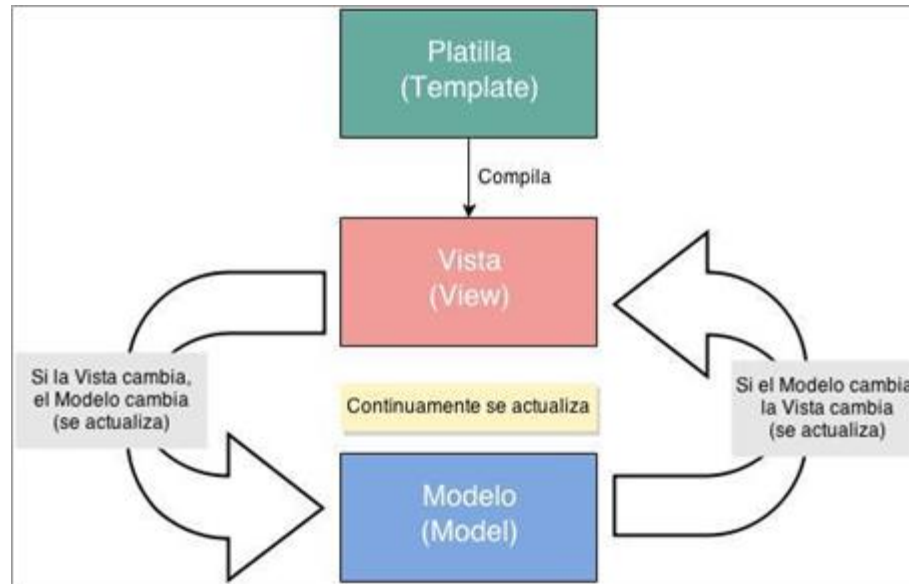
Nuria Lacón

# ¿Qué es Angular JS

- ▶ AngularJS es Javascript. Es un proyecto de código abierto, realizado en Javascript que contiene un conjunto de librerías útiles para el desarrollo de aplicaciones web y propone una serie de patrones de diseño para llevarlas a cabo. En pocas palabras, es lo que se conoce como un framework para el desarrollo, en esta caso sobre el lenguaje Javascript con programación del lado del cliente
- ▶ El objetivo es producir un HTML altamente semántico, es decir, que cuando lo leas entiendas de manera clara qué es lo que hace o para qué sirve cada cosa.
- ▶ Está orientado a aplicaciones web de tipo CRUD (Create - Read - Update - Delete)

# Binding y doble binding

- ▶ El binding sirve para realizar un nexo de unión entre los datos y los distintos contextos de ejecución (scope).
- ▶ Con AngularJS podemos sincronizar el modelo y la vista automáticamente utilizando ciertas directivas del framework. Esta sincronización es bidireccional, es decir, la información se sincroniza tanto si cambia el valor en la vista como si lo hace el valor en el modelo.



# Expresiones

- ▶ Con las expresiones también enriquecemos el HTML, ya que nos permiten colocar cualquier cosa y que AngularJS se encargue de interpretarla y resolverla. Para crear una expresión simplemente la englobas dentro de dobles llaves, de inicio y fin.
- ▶ Otro detalle interesante de las expresiones es la capacidad de formatear la salida, por ejemplo, diciendo que lo que se va a escribir es un número y que deben representarse dos decimales necesariamente:

```
{{ precio | number:2 }}
```

- ▶ Estas expresiones no están pensadas para la lógica de la aplicación así que excepto el operador ternario ( $x ? y : z$ ), no se pueden colocar expresiones de control como bucles o condicionales, sin embargo si podemos llamar a funciones de JavaScript para resolver necesidades complejas.

# Directivas (directive)

- ▶ Las directivas son nuevos "comandos" que vas a incorporar al HTML y los puedes asignar a cualquiera de las etiquetas por medio de atributos. Son como marcas en elementos del DOM de tu página que le indican a AngularJS que tienen que asignarles un comportamiento determinado o incluso transformar ese elemento del DOM o alguno de sus hijos.
- ▶ Cuando se ejecuta una aplicación que trabaja con Angular, existe un "HTML Compiler" (Compilador HTML) que se encarga de recorrer el documento y localizar las directivas que hayas colocado dentro del código HTML, para ejecutar aquellos comportamientos asociados a esas directivas.
- ▶ AngularJS nos trae una serie de directivas "de fábrica" que nos sirven para hacer cosas habituales, así como tú o terceros desarrolladores pueden crear sus propias directivas para enriquecer el framework.

## Using Directives and Data Binding Syntax

```
<!DOCTYPE html>
```

```
<html ng-app>
```

```
<head>
```

```
  <title></title>
```

```
</head>
```

```
<body>
```

```
  <div class="container">
```

```
    Name: <input type="text" ng-model="name" /> {{ name }}
```

```
  </div>
```

```
  <script src="Scripts/angular.js"></script>
```

```
</body>
```

```
</html>
```

Directive

Directive

Data Binding  
Expression

# Directivas (directive)

- ▶ **Directiva ngApp (ng-app):** Esta es la marca que indica el elemento raíz de tu aplicación. Se coloca como atributo en la etiqueta que deseas que sea la raíz. Es una directiva que autoarranca la aplicación web AngularJS. Lo más común es ponerlo al principio de tu documento HTML, en la etiqueta HTML o BODY.
- ▶ **Directiva ngModel (ng-model):** La directiva ngModel informa al compilador HTML de AngularJS que estás declarando una variable de tu modelo. Las puedes usar dentro de campos INPUT, SELECT, TEXTAREA (o controles de formulario personalizados).

Técnicamente, lo que haces con ngModel es crear una propiedad dentro del "scope" (tu modelo) cuyo valor tendrá aquello que se escriba en el campo de texto. Gracias al "binding" cuando modifiques ese valor en el scope por medio de Javascript, también se modificará lo que haya escrito en el campo de texto.

# Directivas (directive)

- ▶ **Directiva ngInit (ng-init):** Esta directiva nos sirve para inicializar datos en nuestra aplicación, por medio de expresiones que se evaluarán en el contexto actual donde hayan sido definidas. Así, de manera general, podemos crear variables en el "scope", inicializarlas con valores, etc. para que en el momento que las vayamos a necesitar estén cargadas con los datos que necesitamos.
- ▶ **Directiva ngRepeat (ng-repeat):** Esta directiva te sirve para implementar una repetición (un bucle). Es usada para repetir un grupo de etiquetas una serie de veces. Al implementar la directiva en tu HTML tienes que decirle sobre qué estructura se va a iterar.
- ▶ **Directiva ngClick (ng-click):** Esta directiva es utilizada para especificar un evento click. En ella pondremos el código (mejor dicho la expresión) que se debe ejecutar cuando se produzca un clic sobre el elemento donde se ha colocado la directiva.



# Directivas (directive)

- Un ejemplo con ng-repeat y con ng-init:

```
<p ng-repeat="elemento in miColeccion" ng-init="paso=$index;">  
  Elemento con id {{paso}}: <span>{{elemento}}</span>  
</p>
```

- La directiva ng-repeat maneja una serie de propiedades especiales que puedes inicializar para el contexto propio de cada repetición. Para inicializarlas usamos la directiva ng-init indicando los nombres de las variables donde vamos a guardar esas propiedades. Angular te proporciona otras propiedades útiles como \$first (que valdrá true en caso que sea la primera iteración) o \$last (true solo para la última iteración)
- Ver ejemplo 01\_Directivas.html

# Controladores (Controller)

- ▶ Los controladores nos permiten mediante programación implementar la lógica de la presentación en AngularJS. En ellos podemos mantener el código necesario para inicializar una aplicación, gestionar los eventos, etc.
- ▶ En líneas generales podemos entender que los controladores nos sirven para separar ciertas partes del código de una aplicación y para que el HTML utilizado para la presentación no se mezcle con el Javascript para darle vida.
- ▶ Un controlador puede ser agregado al DOM mediante la directiva `ngController` y a partir de entonces tendremos disponible en esa etiqueta (y todas sus hijas) los datos necesarios en la presentación de la vista, asociados al modelo o scope (MVC).

# Controladores (Controller)

- ▶ El "scope" es la pieza más importante del motor de AngularJS y es donde están los datos que se tienen que manejar dentro de la parte de presentación.
- ▶ Es un gran contenedor de datos, que transporta y hace visible la información necesaria para implementar la aplicación, desde el controlador a la vista y desde la vista al controlador.
- ▶ Desde los controladores vamos a ser capaces de trabajar con el scope de una manera minuciosa, agregando o modificando información según lo requiera nuestra aplicación.

# Controladores (Controller)

- El código necesario para crear un controlador en AngularJS tendrá este aspecto:

```
var app = angular.module("miapp", []);

app.controller("miappCtrl", function(){
    var scope = this;
    scope.datoScope = "valor";
    scope.metodoScope = function(){
        scope.datoScope = "otra cosa";
    }
});
```

# Módulos (Module)

- ▶ Los módulos vienen a ser contenedores de diferentes partes de nuestra aplicación. Podemos definir la cantidad de módulos que nos sean necesarios para desacoplar totalmente el código, sea por características, por funcionalidad, por componente reusable, etc.
- ▶ Es bueno tener en cuenta que cuanto más desacoplado tengamos nuestro código será mucho más fácil mantenerlo y escalarlo.
- ▶ Para declarar un módulo usaremos la sintaxis:  

```
angular.module('nuevaApp', []);
```
- ▶ El primer argumento es el nombre del módulo. En el segundo parámetro se pueden indicar una serie de módulos adicionales, separados por comas, que serían las dependencias.
- ▶ El nombre del módulo se deberá indicar en la directiva ng-app.

# Filtros y ordenación de elementos

- ▶ Los filters son los encargados de procesar la información antes de mostrarla en pantalla. Permiten hacer muchas cosas, pero lo más habitual es utilizarlos para **modificar los valores a presentar o aplicarles formato**.

- ▶ La utilización de los mismos es similar a los Pipeline de Unix:

```
{{ expresion | filtro }}
```

Donde expresion puede ser cualquier tipo de expresión de AngularJS, como una variable del \$scope, y filtro el nombre del filtro que le queremos aplicar a la expresión.

- ▶ Ver ejemplo [02\\_filtros.html](#)

# Filtros y ordenación de elementos

En angular existe ya algunos filtros predefinidos:

- ▶ **Formatear valores**
  - ▶ `currency`: formatea el número de entrada como moneda
  - ▶ `number`: formatea un número, pudiendo especificar determinado número de decimales
  - ▶ `date`: formatea fechas según se defina
  - ▶ `json`: devuelve a partir de un objeto javascript con su equivalente json como string
  - ▶ `lowercase`: pasa el texto a minúsculas
  - ▶ `uppercase`: pasa el texto a mayúsculas
- ▶ **Filtro o reordenación de colecciones**
  - ▶ `filter`: sirve para filtrar colecciones y solo devolver elementos que cumplan determinada condición
  - ▶ `limitTo`: devuelve los primeros o últimos x elementos de un array o string
  - ▶ `orderBy`: ordena un array

# Filtros y ordenación de elementos

## Filtros personalizado en búsquedas:

- ▶ El filtrado nos sirve para hacer una búsqueda dentro de los elementos de la colección que tenemos en un array o en un objeto en nuestro modelo. Se indica con la palabra **filter**, seguida por ":" y la cadena o variable donde está la cadena que nos sirve para filtrar.
- ▶ La ordenación usaremos la palabra **orderBy**, seguida por ":" y el campo sobre el que se debe ordenar. Opcionalmente colocamos después si el orden es ascendente o descendente.



# Factorías (Factory)

- ▶ Las factorías son como contenedores de código. Son un tipo de servicio con el que podemos implementar librerías de funciones o almacenar datos.
- ▶ Siempre van a devolvernos un dato, de cualquier tipo. Lo común es que nos devuelvan un objeto de Javascript donde podremos encontrar datos (propiedades) y operaciones (métodos).
- ▶ Con diferencia de los controladores, las factorías tienen la característica de ser instanciados una única vez dentro de las aplicaciones, por lo que no pierden su estado. Por tanto, son un buen candidato para almacenar datos en nuestra aplicación que queramos usar a lo largo de varios controladores, sin que se inicialicen de nuevo cuando se cambia de vista.