CURSO DESAROLLO WEB CON MEAN

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
gcaballero@pronoide.es

AngularJS 1

INDICE

¿Qué es AngularJS?

- 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

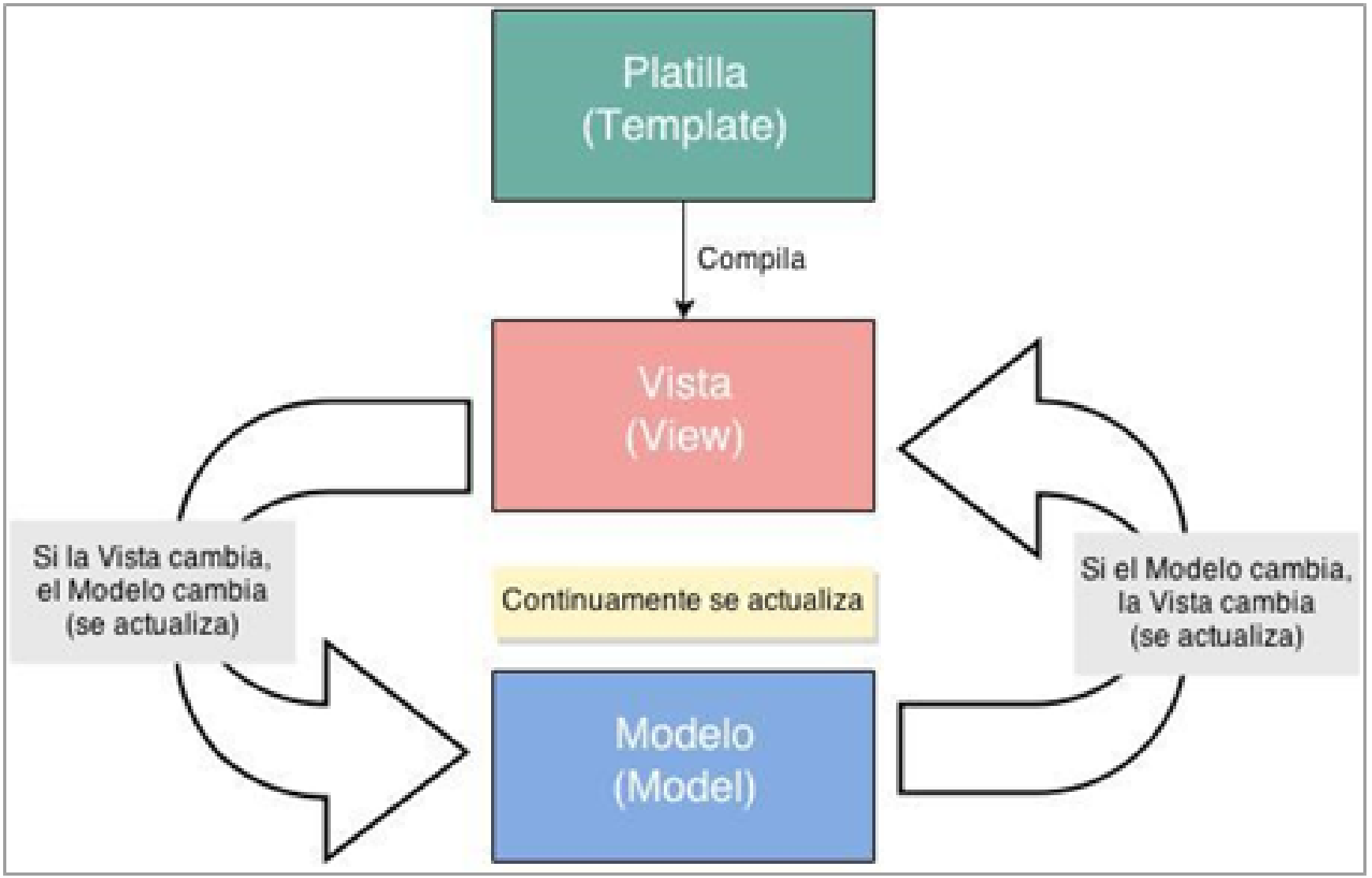
¿Qué es AngularJS?

- 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.

Binding y doble binding



Expresiones

- Con las expresiones 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.

{{ Expresión, variable...}}

Expresiones

- 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

- 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.

Directivas

- 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.

Directivas

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

- **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.

```
1 <html ng-app="app">
2   <head></head>
3   <body></body>
4 </html>
```

Directivas

- **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

- **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 vayas a necesitar estén cargadas con los datos que necesitas.

Directivas

- **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.

```
<p ng-repeat="elemento in miColeccion"
   ng-init="paso=$index;">

      Elemento con id {{paso}}: <span>{{elemento}}</span>

</p>
```

Directivas

- **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.

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.

Filtros y ordenación de elementos

En angular existen ya 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

Filtros y ordenación de elementos

En angular existen ya filtros predefinidos:

- 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.

Módulos

- 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.

Módulos

- 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**.

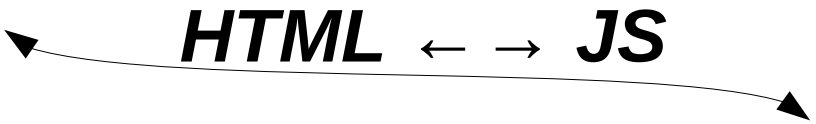
Controladores

- 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.

Controladores

- Un controlador puede ser agregado al DOM mediante la directiva ngController
- 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).

```
<div ng-controller="AppCtrl">
  {{ someValue }}
</div>
```




```
$scope.someValue = 'Hello';
```

Controladores

- 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";
    }
});
```

Factorías

- 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).

Factorías

- 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.
- Las factorías son como contenedores de código que podemos usar en nuestros sitios desarrollados con AngularJS.
- Son un tipo de servicio, "service" en Angular, con el que podemos implementar librerías de funciones o almacenar datos.

Factorías

- 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.
- Angular consigue ese comportamiento usando el patrón "Singleton" que básicamente quiere decir que, cada vez que se necesite un objeto de ese tipo, se enviará la misma instancia de ese objeto en lugar de volver a instanciar un ejemplar

Factorías

Nota:

- El patrón "Singleton" no es algo específico de AngularJS, en realidad es un patrón general de programación orientada a objetos.
- Así como las factorías en líneas generales también son un conocido patrón de diseño de software que se usa en el desarrollo de aplicaciones web y aplicaciones tradicionales orientadas a objetos.

Factorías

Implementación

- Primero crear módulo:

```
angular.module("app", ["ngRoute"])
```

- El mecanismo para crear la factoría es el mismo que hacemos para crear los controladores.
- Para crear el controlador usas el método **controller()** y para la factoría el método **factory().**

Factorías

Implementación

- Sobre el objeto que devuelve esa operación crearemos las factorías

```
.factory("descargasFactory", function(){
    var descargasRealizadas = ["Manual de Javascript", "Manual de jQuery",

    var interfaz = {
        nombre: "Manolo",
        getDescargas: function(){
            return descargasRealizadas;
        },
        nuevaDescarga: function(descarga){
            descargasRealizadas.push(descarga);
        }
    }
    return interfaz;
})
```

Factorías

Implementación

- Esta factoría se llama "descargasFactory".
- El nombre lo hemos definido en la llamada al método factory.
- Este nombre luego se usará al inyectar la dependencia de esta factoría en los controladores.

Factorías

Detalles interesantes

- Entran en juego diversos conceptos de la programación orientada a objetos en Javascript.
- Lo más destacado sobre las factorías en AngularJS lo encuentras en la última línea:

"return interfaz;"

- Todas las factorías deben devolver algo.
- Lo que sea, aunque lo habitual como dijimos es devolver un objeto.
- Por definición debe de ser así en AngularJS.

Factorías

Detalles interesantes

- Aquello que devuelves es lo que se conoce desde fuera de la factoría.
- Por decirlo de otra manera, es la interfaz pública de uso de esa factoría.
- Por eso hemos llamado a la variable que devolvemos en el return "interfaz", porque es la serie de propiedades y métodos que estás haciendo público para todo el mundo que use esa factoría.
- Lógicamente, esa "interfaz" no es más que una manera nuestra de llamar a la variable y tú usarás la que quieras.

Factorías

Detalles interesantes

- La variable "descargasRealizadas" es privada a la factoría, pues no se devuelve en la interfaz.
- Por tanto, ese array no podrá ser accesible desde fuera de la factoría.
- Podemos entenderlo como una propiedad privada.

Factorías

Detalles interesantes

- Para acceder al array "descargasRealizadas" se hará uso de los métodos definidos en "interfaz":

  getDescargas() y **nuevaDescarga()**

- Esos métodos son públicos, por haberlos definido en la interfaz que devolvemos en la función de la factoría y se podrán acceder desde cualquier lugar donde tengamos disponible la factoría.

Factorías

Detalles interesantes

- Sin embargo no todos los datos que vamos a manejar en las factorías necesitamos hacerlos privados.
- En concreto encontrarás, la propiedad "nombre" que está dentro de nuestra interfaz y por lo tanto es pública y podrá accederse tal cual desde fuera de la factoría.

Factorías

Usar una factoría

- El procedimiento es tan simple como, una vez definida, inyectarla en el controlador donde la queremos usar.
- Usamos el sistema de inyección de dependencias.
- Al crear la función del controlador debemos definir un parámetro que se llame exactamente igual al nombre que le has dado en la factoría.
- En este caso el parámetro que inyectamos en el controlador se llama "descargasFactory" pues así habíamos llamado a la factoría al crearla.

Factorías

Usar una factoría

```
.controller("appCtrl", function(descargasFactory){
    var vm = this;

    vm.nombre = descargasFactory.nombre;
    vm.descargas = descargasFactory.getDescargas();
    vm.funciones = {
        guardarNombre: function(){
            descargasFactory.nombre = vm.nombre;
        },
        agregarDescarga: function(){
            descargasFactory.nuevaDescarga(vm.nombreNuevaDescarga);
            vm.mensaje = "Descarga agregada";
        },
        borrarMensaje: function(){
            vm.mensaje = "";
        }
    }
});
```

Factorías

Usar una factoría

- Dentro de nuestro controlador la variable descargasFactory que recibimos como parámetro contiene todos los datos y funciones que hemos definido en la interfaz pública de nuestra factoría.
  - **descargasFactory.nombre** contendrá la propiedad "nombre" definida en la factory.
  - **descargasFactory.nuevaDescarga()** o **descargasFactory.getDescargas()** serán llamadas a los métodos que habíamos definido en la factoría.

http://www.desarrolloweb.com/articulos/factorias-factory-angularjs.html

Servicios

- Los "services" en AngularJS incluyen tanto factorías como servicios
- Por ejemplo, cuando estás haciendo Ajax, por los métodos que hemos conocido hasta el momento, usamos $http.
- Éste no es más que un service de Angular que engloba toda la funcionalidad necesaria para realizar solicitudes asíncronas a un servidor.

Servicios

- Por tanto, algo como Ajax, que se supone que puedes querer realizar a lo largo de tu aplicación en varias partes del código, se ha separado a un "servicio".
- Esto quiere decir que, cuando quieras hacer Ajax, tendrás que usar el código del "service" $http.
- Los servicios y factorías que desees usar en tus controladores o módulos deberás inyectarlos como has visto hacer en diversas partes de nuestros ejemplos.