# AngularJS

Juanda

Abril 2015

# Contents

# Chapter 1

# Introducción

## 1.1 Qué es AngularJS

Es la librería más utilizada de JavaScript.

Está mantenida por Google

Uno de los frameworks más usados (tanto en cliente como servidor) en la actualidad.

Muy adecuado para el desarrollo de un SPA (Single Page Application)

## 1.2 Entorno de trabajo

Navegador Web Chrome con extensión **Batarang AngularJS** para debug

Editor de código Sublime con extensión AngularJS

nodejs

Hay varios generadores interesantes para automatizar nuestras tareas de desarrollo en Angular mediante Yeoman

Es habitual utilizar un Mean Stack para desarrollo

- Mean hace referencia al uso de las tecnologías MongoDB, Express, AngularJS y Node
- Es probable que sustituya a los sistemas WAMP/LAMP

Utilizaremos Bootstrap para el diseño del frontend.

La unión del backend y el frontend se hace mediante la creación de una arquitectura API REST

Un servidor web

## 1.3 Servidor web: nodejs

Es necesario para que las peticiones ajax funcionen

Lo crearemos a mano mediante node (fichero *server.js*):

```
var connect = require('connect'),
     serveStatic = require('serve-static');
var app = connect();
app.use(serveStatic('.'));
app.listen(5000);
```

Para que funcione debemos instalar las dependencias del fichero:

```
npm install connect
npm install serve-static
```

Ejecutamos el servidor web mediante:

```
node server.js
```

Nuestro servidor web servirá ficheros por el puerto 5000

Servirá cualquier contenido que se encuentre bajo su directorio (*serveStatic('.')*)

```
http://localhost:5000
```

## 1.4 Servidor web: python

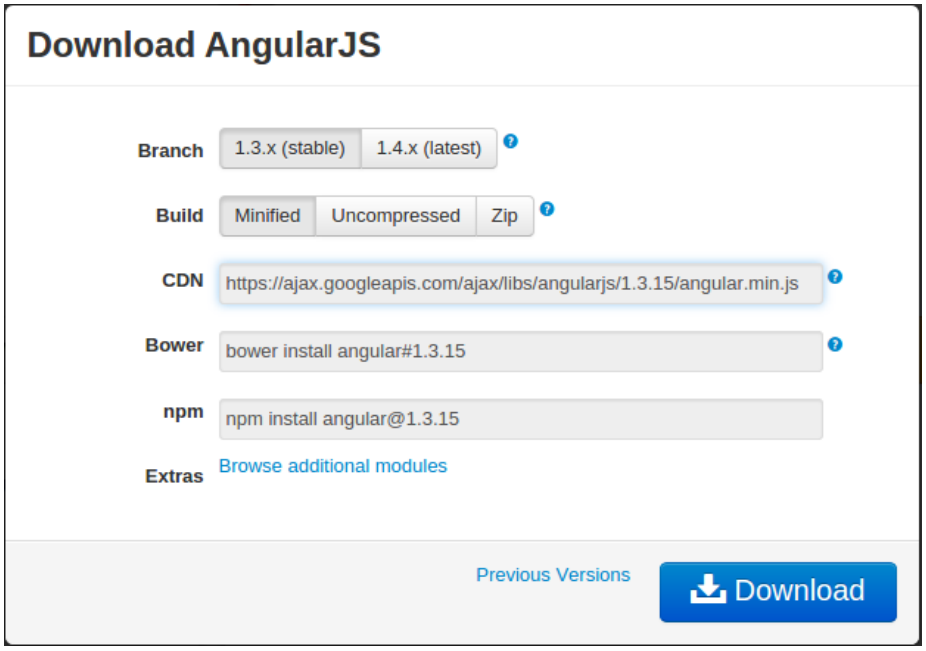
Otra opción sería usar Python:

```
python -m SimpleHTTPServer 5000
```

## 1.5 Librerías de AngularJS

Vamos a la web de AngularJS: https://www.angularjs.org/

Pulsamos en la opción de Download




Podemos descargar angularJS directamente mediante gestores de paquetes: bower o npm

Podemos utilizar un CDN: https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js

## 1.6 Librerías extra de AngularJS

En la misma URL anterior está la opción de descargar otras librerías útiles (opción extras):

- **angular-touch.js**: para soportar eventos táctiles
- **angular-animate.js**: para animaciones
- **angular-route.js**: para enrutamiento de URL's
- **angular-locale-xx-xx.js**: Para localización de nuestra aplicación

El repositorio del código de Angular se encuentra en https://code.angularjs.org/

# Chapter 2

# Arquitectura de una aplicación Angular

## 2.1 Arquitectura de Angular

Angular se conoce como: *Superheroic JavaScript MVW Framework*

MVW son las siglas de

- Modelo
- Vista
- Whatever

La arquitecturas más conocidas son las MVC, ¡pero en servidor!

AngularJS no implementa MVC en la forma tradicional, se parece a una arquitectura MVVM (Model-View-ViewModel)

Como es una arquitectura versátil, para denominarla le añadieron el whatever :-)

## 2.2 Arquitectura MVC

MVC es un **patrón de diseño** para la arquitectura de las aplicaciones.

Separa los contenidos de nuestra aplicación en 3 partes bien diferenciadas:

- **Modelo**: Recoger los datos, normalmente de una bbdd y los almacena en objetos para su uso.

- **Vista**: Muestrar la información al usuario
- **Controlador**:
  - Recoge la lógica de la aplicación, son las funciones de toda la vida
  - Actúa como enlace: Recoge los datos del modelo, los procesa si es necesario y se los pasa a la vista para su renderización en el navegador.

## 2.3 Pros&Cons arquitectura MVC

Tiene muchas ventajas, al estar el código desacoplado:

- El código está ordenado
- Puedes cambiar el modelo sin tener que cambiar la vista.
- Reutilizar componentes
- Las pruebas unitarias son más sencillas

Alguna desventaja:

- Requiere cierta abstracción
- Coste de aprendizaje frente a las fechas de entrega del software

## 2.4 ¿Dónde pongo mi código?

Se sigue la norma de **Fat model, skinny controller**:

- Si tienes dudas, pon el código en el modelo
- De cualquier forma cuanto menos código en cada parte, más reusable se vuelve nuestra aplicación

Al "partirse el código" necesitas moverte rápidamente entre los distintos ficheros del código

- Utiliza un editor de código que te lo permita mediante teclas rápidas

## 2.5 Arquitectura en el cliente

Traslado de la **lógica del negocio al cliente**

Los **tiempos de respuesta** se vuelven similares a los de una aplicación de escritorio.

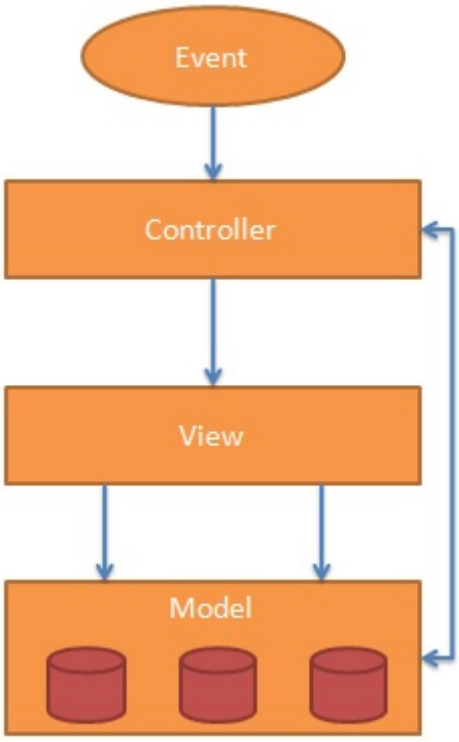
Necesitaremos una API REST en el servidor para que nuestra aplicación funcione

El desarrollo de la aplicación cambia:

- Desarrolladores de frontend
- Desarrolladores de backend
- Interfaz en común (API) que se simula para evitar malentendidos.

## 2.6 Acceso al Modelo

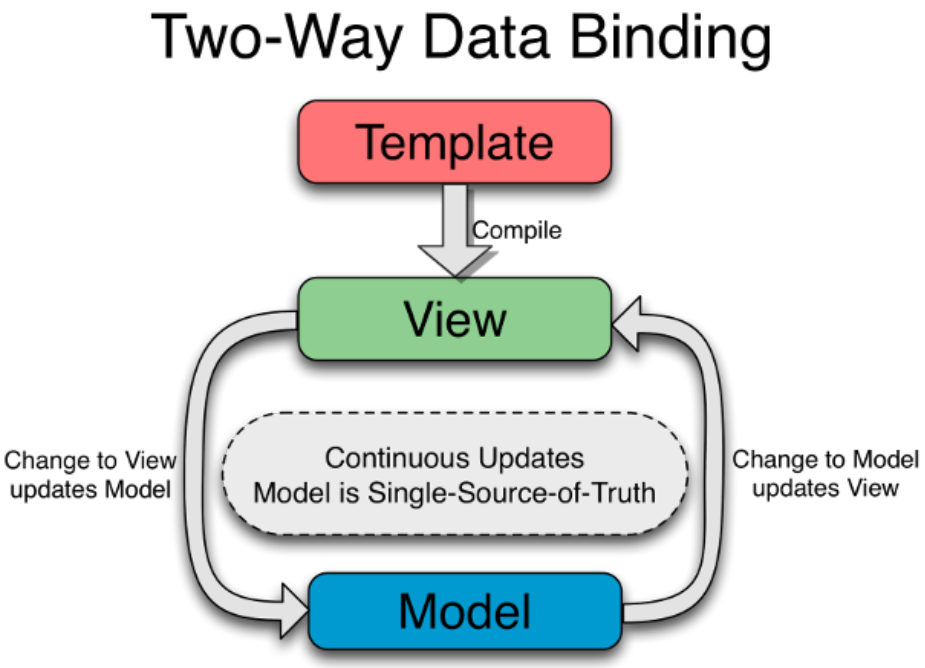
Al modelo se puede acceder tanto desde la vista como desde el controlador.



## 2.7 Arquitectura MVW - Databinding

**Databinding**: La vista se actualiza si hay cambios en el modelo

**Two way databinding**: El modelo se actualiza si hay cambios en la vista




## 2.8 Ejemplo aplicación con jQuery

Con jQuery:

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></s
    <script>
        $(document).ready(function() {
            var $nombre = $('#nombre');
            $nombre.keyup(function() {
                $('#saludo').text('¡Hola ' + $nombre.val() + '!');
            });
        });
```

```
    </script>
</head>
<body>
    <input id="nombre" type="text">
    <h2 id="saludo"></h2>
</body>
</html>
```

## 2.9 Ejemplo en AngularJS mediante databinding

El modelo se declará mediante el atributo **ng-model**.

Si el modelo cambia, se actualizará la vista.

```
<!doctype html>
<html ng-app>
<head>
    <meta charset="UTF-8">
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.0/angular.min.js"></script>
</head>
<body>
    <input type="text" ng-model="nombre">
    <h2>Hola {{nombre} }</h2>
</body>
</html>
```

## 2.10 Controlador

En el código anterior se puede echar de menos un controller:

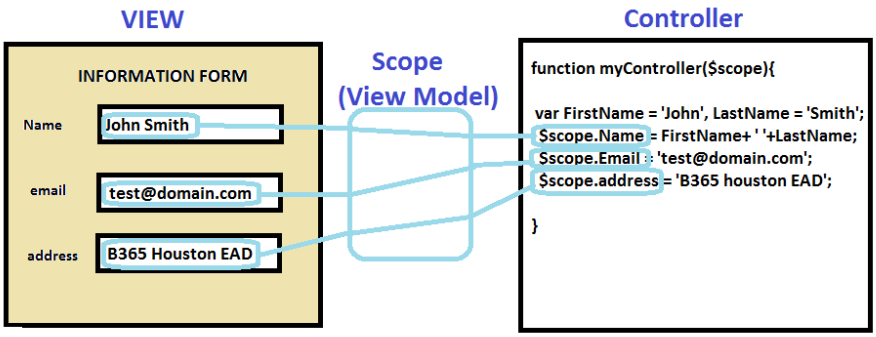
- ¿Cómo se hace persistente el cambio en una bbdd?
  - Necesitaremos hacer una llamada al servidor API REST
- ¿Cómo se obtienen los datos para mostrar?
  - Necesitaremos hacer una llamada al servidor API REST
- ¿Cómo modificamos los datos?
  - Código en la vista mediante expresiones (más sucio si hay muchos cambios)
  - Código js puro y duro en el controlador (separamos la lógica de la presentación de datos)

## 2.11 Scope o View Model

La vista desconoce la lógica del controlador o sus datos.

El controlador desconoce la vista

El scope es el objeto con el que el controlador le pasa los datos a la vista.



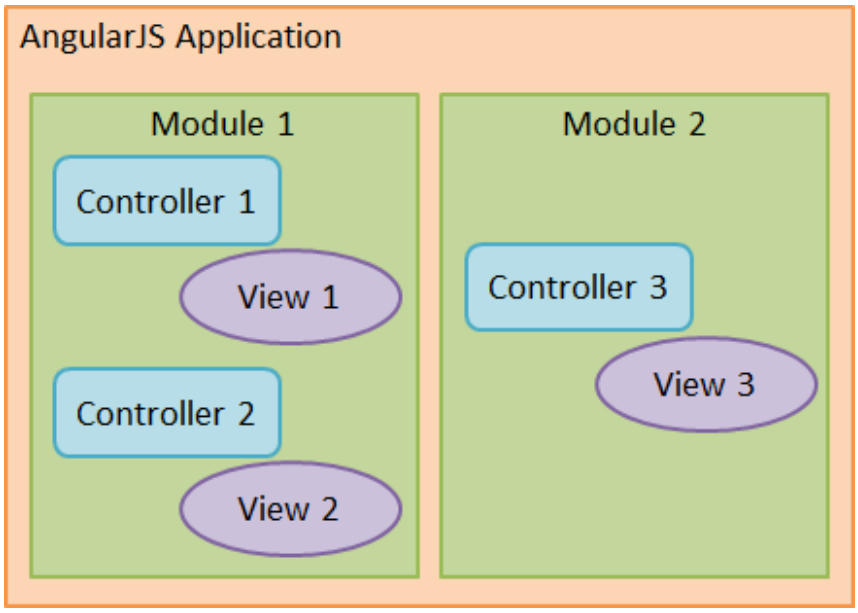

## 2.12 Modules

Los modules son las partes lógicas en las que puedes dividir tu aplicación

Cada módulo tendrá uno o más controladores

Cada módulo tendrá una o más vistas

- Se definen mediante **routing**, se verá más adelante




## 2.13 Ejemplo: módulo y controlador

Tenemos un controllador con una lista de jugadores de futbol

Queremos mostrarlos en la vista

```
var myApp = angular.module('myApp',[]);
myApp.controller('FubtolistasController', ['$scope', function($scope) {
    $scope.Futbolistas = [
        {nombre: 'Lionel', apellido:'Messi', equipoActual: 'FC Barcelona' },
        {nombre: 'Cristino', apellido:'Ronaldo', equipoActual: 'Real Madrid' },
        {nombre: 'Fernando', apellido:'Torres', equipoActual: 'Atlético' },
    ];
}]);
```

Definimos nuestro módulo myApp

Asociamos un controlador FutbolistasController

## 2.14 Ejemplo: html

Asociamos el controlador a una etiqueta de html

Mediante la directiva ng-repeat y usando expresiones, recorremos el array de datos

```
<div ng-controller="FutbolistasController">
    <h2>Lista de futbolistas</h2>
    <ul>
        <li ng-repeat="futbolista in Futbolistas">
            { {futbolista.nombre}} {{futbolista.apellido} } juega en el { {futbolista.e
        </li>
    </ul>
</div>
```

# Chapter 3

# Mi primera aplicación

## 3.1 Crear un boilerplate

La plantilla de mi documento html podría ser la siguiente:

```
<!DOCTYPE html>
<html lang="es" ng-app="">
<head>
    <meta charset="UTF-8">
    <title>Mi primera aplicación</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"></script>
</head>
<body>
    <p>Si a continuación ves un 7 en el navegador, tienes todo bien configurado :-) </p>
    <p>{{5+2}}</p>
</body>
</html>
```

## 3.2 Prueba del boilerplate

El contenido entre llaves indica a AngularJS que es una expresión que debe evaluar

El navegador debería mostrar un 7 si ejecutamos el documento anterior:

- Levantamos el servidor web

```
node server.js
```

Escribimos en el navegador *http://localhost:5000/documento.html

## 3.3 Análisis de la estructura

En el documento html anterior aparecen dos cosas “raras”:

- Un atributo **ng-app**
- Una **expresión** que AngularJS calcula

Observa que si ponemos el script de angularJS al final del body, se llega a ver la expresión antes de su evaluación.

El documento no es válido según la W3C

## 3.4 ng-app

ng-app es una **directiva de angular**

También es un atributo html, pero desconocido:

- Hace que el documento no sea válido según la w3c.
- HTML5 permite atributos extendidos (hechos por ti), a partir de data-.
- Los atributos de AngularJS comienzan con ng-, pero también se puede usar data-ng-, para hacer que su página esté validada con HTML5

El atributo ng-app es necesario:

- Indica que lo que hay dentro de sus etiquetas es código de una aplicación Angular
- Si no lo ponemos la expresión no se evaluará

## 3.5 Limitar nuestra aplicación angular

Mediante ng-app podemos acotar nuestra aplicación angular a una parte del html

La segunda expresión no se evaluará:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
```

```
    <title>Mi primera aplicación</title>
</head>
<body>
    <div ng-app="">
        <p>{{5+2}}</p>
    </div>
    <p>{{5+2}}</p>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"></script>
</body>
</html>
```

## 3.6 Directivas

Las directivas en AngularJS son **nuevos atributos o etiquetas HTML** que extienden el documento html

La directiva ng-app inicializa una aplicación AngularJS

Hay muchas más, **AngularJS extiende HTML a través de sus directivas**

- Por ejemplo podemos utilizar ng-init para inicializar datos (normalmente no lo haremos así, utilizaremos el controller)
- Esos datos luego los podremos manajar a través de una expresión

```
<!DOCTYPE html>
<html lang="es" ng-app="">
<head>
    <meta charset="UTF-8">
    <title>Mi primera aplicación</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"></script>
</head>
<body>
    <p ng-init="numero=5">{{numero+2}}</p>
</body>
</html>
```

## 3.7 Controllers

Un controller se asocia a un módulo determinado

Se escriben con el sufijo Controller

Es el sitio donde definimos casí toda la lógica de nuestra aplicación

```
var app = angular.module('saludar', []);
app.controller('mensajesController', ['$scope', function($scope){
    //aquí iría el código javascript
}]);
```

## 3.8 Aplicación Hola Mundo

Definimos el módulo que va a arrancar nuestra aplicación (ng-app) y la acotamos en el documento.

Mediante la directiva ng-controller asociamos un controlador a una parte específica del código

Accedemos a los datos del controllador mediante una expresión

Cargamos las librerías de angular

Cargamos el script en el que definimos nuestro módulo y los controladores (podrían ser varios ficheros)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Saludar</title>
</head>
<body ng-app="saludar">
    <div ng-controller="mensajesController">
        <p>{{saludo}}</p>
    </div>
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.0/angular.min.js"></scri
    <script src="saludar.js"></script>
</body>
</html>
```

## 3.9 Controller Hola Mundo

El nombre del módulo debe coincidir con el valor de la directiva ng-app en el html

El nombre del controlador debe coincidir con el valor de la directiva ng-controller en el html

```
var app = angular.module('saludar', []);
app.controller('mensajesController', ['$scope', function($scope){
    $scope.saludo = "¡Hola Mundo!"
}]);
```

## 3.10 Aplicación: sintaxis nueva

Podemos ver también esta otra sintaxis:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Saludar</title>
</head>
<body ng-app="saludar">
    <div ng-controller="mensajesController as mensajes">
    <p>{{mensajes.saludo}}</p>
    </div>
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.0/angular.min.js"></script>
    <script src="saludar2.js"></script>
    </body>
</html>
```

## 3.11 Sintaxis js

Asociamos la variable al controlador por medio de this

Podemos acceder directamente desde el controlador (sin usar $scope)

Funciona desde la versión 1.3 de Angular

```
var app = angular.module('saludar', []);
app.controller('mensajesController', function(){
    this.saludo="¡Hola Mundo!";
});
```

## 3.12 Built-in directives

Angular viene con muchas directivas que podemos utilizar:

- ng-show

- ng-hide
- ng-repeat
- ng-click
- etc.

Hagamos ahora un Hola Mundo un poco más elaborado con estos requerimientos:

- Con databinding, como en el caso anterior.
- Se debe poder apreciar el two-way-databinding
- Nos puede ayudar utilizar alguna directiva de las anteriores, por ejemplo ng-click.

## 3.13 Hola Mundo con ng-click

Creamos un div que será donde se podrá utilizar el controller (o su $scope)

La función saludar() se ejecutará al hacer clic en el elemento h2 y debe estar definida en el $scope.

```
<!doctype html>
<html ng-app="holaMundoApp">
<head>
    <meta charset="UTF-8">
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.0/angular.min.js"></scri
    <script src="holamundo.js"></script>
</head>
<body>
    <div ng-controller="holaMundoController">
        <input type="text" ng-model="nombre">
        <h2 ng-click="saludar()">Hola {{nombre}}</h2>
    </div>
</body>
</html>
```

## 3.14 Hola Mundo con ng-click: js

En la función saludar() hay un alert de modo que se aprecia que la variable nombre del js corresponde con la expresión {{nombre}} de la vista

```
var miApp=angular.module('holaMundoApp', []);
miApp.controller('holaMundoController', ['$scope', function($scope){
    $scope.nombre = ' Mundo!';
    $scope.saludar = function() {
        alert ($scope.nombre);
    };
}]);
```

# Chapter 4

# Formularios

## 4.1 Validación de formularios

Angular dispone de varias propiedades (booleanas) y clases asociadas para tratar con formularios

- Propiedad $valid y class="ng-valid" para formularios válidos
- Propiedad $invalid y class="ng-invalid" para formularios no válidos
- Propiedad $pristine y class="ng-pristine" para campos sin rellenar
- Propiedad $dirty y class="ng-dirty" para campos "usados".

Acceso a las propiedades de los formularios:

- Formulario:

```
<nombre formulario>.<propiedad angular>
```

- Campo del formulario:

```
<nombre formulario>.<nombre input>.<propiedad angular>
```

## 4.2 Requerimientos ejemplo

Utilizaremos Bootstrap como css

Reglas de validación

- nombre es requerido
- nombre de usuario tiene un mínimo de 3 caracteres y un máximo de 8. No es requerido
- email no es requerido, pero si se inserta, debe ser válido

## 4.3 Código base

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <link href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css" rel="s
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.0/angular.min.js"></scri
    <script src="formulario.js"></script>
</head>

<body ng-app="validacionApp" ng-controller="mainController">
    <div class="container">
        <div class="col-sm-8 col-sm-offset-2">
            <div class="page-header">
                <h1>Validación formulario mediante AngularJS</h1></div>
            <!-- al hacer el submit envíaremos en una variable si el formulario es váli
                haciendo referencia al <form name>.<propiedad> -->
            <form name="formulario" ng-submit="submitForm(formulario.$valid)" novalidat
                <div class="form-group">
                    <label>Nombre</label>
                    <input type="text" name="nombre" class="form-control" ng-model="usu
                </div>
                <div class="form-group">
                    <label>Nombre de Usuario</label>
                    <input type="text" name="nombreusuario" class="form-control" ng-moo
                </div>
                <div class="form-group">
                    <label>Email</label>
                    <input type="email" name="email" class="form-control" ng-model="usu
                </div>
                <button type="submit" class="btn btn-primary">Enviar</button>
            </form>
        </div>
    </div>
</body>
```

```
</html>
```

## 4.4 Deshabilitar botón de envío

Utilizamos la directiva ng-disabled de Angular para controlar el estado del botón de envío del formulario

La propiedad $invalid del formulario será true si algún campo no es válido.

```
<button type="submit" class="btn btn-primary" ng-disabled="formulario.$invalid">Enviar</button>
```

## 4.5 Mensajes de ayuda

Podemos sacar mensajes de ayuda

```
<p ng-show="formulario.nombre.$invalid && !formulario.nombre.$pristine" class="help-block">El nom
<p ng-show="formulario.nombreusuario.$error.minlength" class="help-block">El nombre de usuario es
<p ng-show="formulario.nombreusuario.$error.maxlength" class="help-block">El nombre de usuario es
<p ng-show="formulario.email.$invalid && !formulario.email.$pristine" class="help-block">El email
```

## 4.6 Estilos mediante las clases de Angular

Podemos usar las clases que Angular inyecta de forma dinámica para generar estilos:

```
    .ng-valid       {  }
    .ng-invalid     {  }
    .ng-pristine    {  }
    .ng-dirty       {  }
```

También otras reglas más específicas:

```
    .ng-invalid-required        {  }
    .ng-invalid-minlength       {  }
    .ng-valid-max-length        {  }
```

## 4.7 Clase condicional ng-class

Quizá lo más cómodo es aprovechar la clase **has-error** que Bootstrap posee.

Podemos utilizar la directiva ng-class para aplicar la clase anterior a los elementos html:

```
ng-class="{ <clase a aplicar> : <expresión a evaluar > }"
```

Si expresión es true, se aplicará la clase

```
<div class="form-group" ng-class="{ 'has-error' : formulario.nombre.$invalid && !formu
     <label>Nombre</label>
     <input type="text" name="nombre" class="form-control" ng-model="usuario.nombre" r
     <p ng-show="formulario.nombre.$invalid && !formulario.nombre.$pristine" class="he
</div>
```

## 4.8 Eventos para formularios

Disponemos de muchos eventos para asociar a nuestros elementos. Por ejemplo ng-click para un botón:

- ng-click
- ng-dbl-click
- ng-mousedown
- ng-mouseup
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-keydown
- ng-keyup
- ng-keypress
- ng-change

# Chapter 5

# Filters

## 5.1 ¿Para qué sirven?

Los filtros se usan para **modificar los datos**.

Se pueden usar en directivas, servicios y expresiones.

Los más usados son los siguientes:

- **uppercase**: para convertir un texto en mayúsculas
- **lowercase**: para convertir un texto en minúsculas
- **currency**: formatea un texto como precio o importe.
- **filter**: filtra un array para obtener un subconjunto en base a algún criterio
- **orderby**: ordena un array en función de algún criterio

## 5.2 Sintaxis

En las templates de vistas se utilizan mediante una tubería:

```
{{ expression | filter }}
```

Los filtros se pueden anidar: ~ {{ expression | filter1 | filter2 | ... }} ~

Los filtros pueden tener argumentos:

```
{{ expression | filter:argument1:argument2:... }}
```

## 5.3 Ejemplo de uso de filtros:

```
Introduce tu nombre:<input type="text" ng-model="cliente.nombre">
Introduce tu apellido <input type="text" ng-model="cliente.apellido">
Nombre completo: {{cliente.nombreCompleto() | uppercase}}
```

## 5.4 Localización

La salida del filtro siguiente será 1,234.00

```
{{ 1234 | number:2 }}
```

Angular soporta i18n/l10n para los filtros date, number y currency.

Bastará añadir el script correspondiente para que la salida sea la adecuada

```
<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.3.0/i18n/angular-locale_es-es
```

# Chapter 6

# Servicios

## 6.1 ¿Qué es un servicio?

Son funciones JavaScript con una tarea determinada

Siguen un patrón singleton:

- Objeto con una unica instancia
- Mantenimiento y testing sencillo

Extienden la funcionalidad del controlador:

- La función se "externaliza"
- Al definir el controlador se indican que depende de esa función (dependency inyection)

Ejemplos de servicios:

- $http
- $route
- $window
- $location

## 6.2 Dependency Injection

Es un patrón de diseño que:

- Permite que un componente declare las dependencias que va a utilizar
- Evita que las dependencias estén hardcoded en el componente.

Declaramos las dependencias del controlador como parámetros de la función.

```
phonecatApp.controller('PhoneListCtrl', function ($scope, $http) {
  ....
});
```

Si hacemos el minify de nuestros ficheros, $scope cambiará (es una variable) y la dependencia no se encontrará, se debe pasar también como string:

```
phonecatApp.controller('PhoneListCtrl', ['$scope', '$http', function ($scope, $http) {
  ....
]});
```

## 6.3 $http service

Servicio presente en el núcleo de Angular para comunicarse con servidores web remotos de forma asíncrona

- Vía XMLHttpRequest
- Vía JSONP

Para pruebas unitarias se puede utilizar el servicio $httpBackend (módulo ngMock)

Para un nivel de abstracción superior podemos utilizar $resource (módulo ngResource)

## 6.4 $http vía get

```
$http.get('/someUrl').success(successCallback);
$http.post('/someUrl', data).success(successCallback);
```

$http devuelve una promesa.

Podremos utilizar otros métodos como then o error.

Otras opciones

- $http.head
- $http.put
- $http.delete
- $http.jsonp
- $http.patch

# Chapter 7

# Vistas y Rutas

## 7.1 Includes

html no soporta incluir una página html dentro de otra:

- Utilizamos ajax: obtenemos la página embebida y la colocamos en un elemento html a partir de la propiedad innerHTML
- Usando includes en servidor

## 7.2 ng-include

En Angular podemos hacer includes mediante la directiva **ng-include**

Angular espera una variable. Por eso datosclientes.html va entre comillas simples:

```
<div ng-app="" ng-controller="clienteController">
   <div ng-include="'main.htm'"></div>
   <div ng-include="'datoscliente.htm'"></div>
</div>
```

## 7.3 ng-view

ng-include a veces no es suficiente (ni necesario siquiera)

ng-view es totalmente necesario para un SPA

- Se integra con el routing

- Podemos cargar también controladores para cada ng-view

Crearemos un placeholder donde se puede introducir una vista (html o vista mediante ng-template)

```
<div ng-app="mainApp">
...
   <div ng-view></div>

</div>
```

## 7.4 Librería ngRoute

Para cargar un sistema de enrutado necesitamos la librería ngRoute

ngRoute no forma parte del core del Angular (antes sí), así que debemos cargarla:

```
<script src="//ajax.googleapis.com/ajax/libs/angularjs/X.Y.Z/angular-route.js"></script
```

## 7.5 Incluir ngRoute en nuestra aplicación

Debemos incluir la dependencia a nuestra librería

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

Usaremos el servicio $routeProvider que sirve para:

- Añadir rutas a nuestro servicioi $route:

  ```
  .when(path, route);
  ```

- Definir una ruta por defecto:

  ```
  .otherwise(params);
  ```

- Asociar urls con páginas html o ng-templates
- Asociar urls con controladores

## 7.6 Como usar $routeProvider

Crearemos un fichero routes.js que sea el encargado de manejar las rutas de nuestra aplicación

Introduciremos un código similar al siguiente:

```
angular.module('MainApp')
       .config(['$routeProvider',
          function($routeProvider) {
             $routeProvider.
                when('/addStudent', {
                   templateUrl: 'addStudent.htm',
                   controller: 'AddStudentController'
                }).
                when('/viewStudents', {
                   templateUrl: 'viewStudents.htm',
                   controller: 'ViewStudentsController'
                }).
                otherwise({
                   redirectTo: '/addStudent'
                });
          }]);
```

## 7.7 Rutas con parámetros

¿Cómo veríamos un estudiante en partícular?

```
http://www.miapp.com/#/viewStudents/<id>
```

La configuración en nuestro fichero de rutas sería:

```
.when ('/viewStudents/:id', {
    .....
});
```

```
Podríamos pasar también varios parámetros (nombre y apellidos) y parámetros opcionales (mote):
```

.when ('/viewStudents/:nombre/:apellido/:mote?', { ..... }); ~

Y para obtener el valor de algún parámetro de la ruta desde el controlador utilizaremos $routeParams:

```
.controller('ViewStudentsController', function($routeParams) {
  var self = this;
  self.estudiante = $routeParams.nombre + " " + $routeParams.apellido;
});
```

## 7.8 Rutas más avanzadas

Con las rutas anteriores hemos configurado urls del tipo:

```
http://www.miapp.com/#/addStudent
http://www.miapp.com/#/viewStudents
```

Angular coloca el hash por defecto (se podría quitar)

Podríamos tener requerimientos más complejos como:

- vistas anidadas
- urls generadas de forma dinámica

Para estos y otros casos utilizaremos ui-route (módulo de terceros) en vez de ngRoute.

# Chapter 8

# Custom Directives

## 8.1 Componentes en html

Piensa como insertaríamos algún componente javascript en nuestra página web:

- Seleccionaríamos algún elemento html
- Lo convertiríamos en nuestro componente mediante JavaScript

Ejemplo con DataTables:

- Marcaríamos un elemento de tipo table dentro del html con un id (por ejemplo example).
- Seleccionamos el elemento mediante jQuery a partir del id
- DataTable() es la función de JavaScript que inserta el componente

```
$(document).ready(function() {
    $('#example').DataTable();
} );
```

Ejemplo con un datePiker:

```
$('input[type=text]').datePicker()
```

## 8.2 Concepto de Custom Directives

Angular intenta extender el html en base a directivas

- El código se vuelve más legible
- Se pueden crear nuevas directivas

Nuestras nuevas directivas podrían ser un nuevo elemento html:

```
<data-table></data-table>
```

```
<date-picker></date-picker>
```

Nuestras nuevas directivas podrían ser también un nuevo atributo html:

```
<table id="example" data-table/>
```

```
<input type="text" date-picker/>
```

## 8.3 Construir una Custom Directive

**app.directive()** registra la directiva

- Se registran de forma similar a un controroller
- El segundo argumento es la función que define la directiva

**restrict** restringe las formas en que podemos usar la directiva:

- A: atributo
- E: elemento
- C: clase

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

app.directive('HolaMundo', function() {
  return {
      restrict: 'AE',
      replace: 'true',
      template: '<h3>¡Hola Mundo!</h3>'
  };
});
```

El camelCase al pasar a html se debe convertir a un guión o a dos puntos

```
<hola:mundo></hola:mundo>
<hola-mundo></hola-mundo>
<div hola:mundo></div>
<div hola-mundo></div>
```

## 8.4 Por qué usar directivas

html solo nos informa sobre la estructura de nuestra aplicación

Las directivas permiten escribir html que exprese el comportamiento de nuestra aplicación:

```
<aside class="col-sm-3">
  <libro-portada></libro-portada>
  <h4><libro-valoracion></libro-valoracion></h4>
</aside>
<div class="col-sm-9">
  <h4><libro-titulo></libro-titulo></h4>
  <libro-autor></libro-autor>
  <libro-genero></libro-genero>
    ....
</div>
```

## 8.5 Custom directive como include

Cambiaremos template por templateUrl

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

app.directive('HolaMundo', function() {
  return {
      restrict: 'AE',
      replace: 'true',
      templateUrl: 'holamundo.html'
  };
});
```