Expressions i Filtres

Expressions

Introducció de Strings

Amb les expressions també s'enforteix L' HTML, ja que ens permet col·locar qualsevol cosa i que AngularJS s'encarregui d'interpretar-la i solventar-la. Per a crear una expressió simplement la englobes dins de dobles claus, de inici i final.

Ara dins de la aplicació, en qualsevol lloc del codi HTML delimitat per l'etiqueta ón vam posar la directiva ng-app es poden col·locar expressions. En Angular disposem de diferents tipus d'expressions., Veiem un exemple:

```
<h1>{{ 1 + 1}}</h1>
```

Angular quan es posa a executar l'aplicació buscarà expressions d'aquest estil i el que tinguin a dins, si és una cosa que ell pugui avaluar, el resoldrà i el substituirà amb el resultat que correspongui.
Pots provar una altra expressió com aquesta:

```
{{ "Hola " + "DesarrolloWeb" }}
```

A l'executar-lo, AngularJS substituirà aquesta expressió per "Hola Desarrolloweb". Aquestes expressions no m'aporten gran cosa de moment, però després les utilitzarem per a més tipus d'operacions.
El més habitual de les expressions és utilitzar-les per col·locar dades del teu model, escrivint els noms de les "variables" que tinguis en el model. Per exemple, si tens aquest codi HTML, on has definit una dada en el teu model anomenat "valor"

```
<input type="text" ng-model="valor" />
```

Podràs bolcar a la pàgina aquesta dada (el que hagi escrit en el camp INPUT) per mitjà de la següent expressió:

```
{{valor}}
```

Un altre detall interesant de les expressions és la capacitat de formatejar la sortida, per exemple, dient que el que es va a escriure és un número i que han de representar-se dos decimals necessàriament. Veiem ràpidament la sintaxis d'aquesta sortida formatejada, i més endavant la utilitzarem en diversos exemples:

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

Cal dir que aquestes expressions no estan pensades per escriure codi del, o sigui, el que anomenem lògica de l'aplicació. Com dèiem, estan sobre tot pensades per a bolcar la informació que tinguis en el teu model i facilitar el "binding". Per això, excepte el operador ternari (x ? y : z), no es poden col·locar expressions de control como bucles o condicionals. En canvi, des de les expressions també podem cridar a funcions codificades en Javascript (que estan escrites en els controladors, com veurem dintre de poc) per a poder resoldre qualsevol tipus de necessitat més complexa.
Fins aquí hem descrit tots els components d'AngularJS. Ara anem a conèixer altres components de les aplicacions que podem fer amb aquest framework Javascript.

Crear filtres

Els filtres permeten modificar com es mostren les dades en la pàgina HTML. Exemples d'això és com es mostren les dates, els nombres, com ordenar llistes, etc.

Existeixen 2 tipus de filtres segons a que tipus de dades s'apliquen:

- Filtres per escalars: S'apliquen a dades com nombres, dates, text, etc. Però no a la llista de dades
- Filtres per llistes de dades: S'apliquen a llista de dades com arrays

Els filtres s'utilitzen afegint al valor a mostrar el caràcter de canonada i després el nom del filtre i les seves opcions.

```
{{valor | filtro:opciones }}
```

Veiem un exemple:

```
<div>{{importe | number:2}}</div>
```

En aquest exemple al mostrar la variable importe que és de tipus numèric s'aplica un filtre que limita el nombre de decimals a mostrar a 2 decimals.
Es poden aplicar varis filtres diferents concatenant més canonades i el nom del següent filtre. El següent exemple mostra 3 filtres:

{{valor | primerFiltro | segundoFiltro | tercerFiltro }}
El signe dels dos punts i *opcions* només es pensaran si el filtre corresponent necessita de paràmetres per a funcionar. En cas de que hi hagui més d'un paràmetre es separarà per comes.
Un exemple de paràmetre és en el filtre date per especificar el format de la data a mostrar.

Escalars

Aquests filtres s'apliquen per a modificar la forma en la que es mostren dades escalars.
Els diferents tipus de filtres per a escalars que existeixen en AngularJS són:

- number
- date
- currency
- lowercase
- uppercase

number

Aquest filtre s'aplica sobre nombres par a limitar el nº de decimals que es mostren d'aquest nombre. Encara que també canvia el separador de decimals al idioma que aquest estigui utilitzant actualment.

format:

```
{{valor | number:numdecimales }}
```

Veiem ara un exemple:

```
<div>{{importe | number:2}}</div>
```

La variable importe només es mostrarà amb 2 decimals.
Si no s'indica el nombre de decimals, per defecte es mostren 3 decimals.

date

Aquest filtre s'aplica sobre dates per a mostrar un format concret.

format:

```
{{valor | date:'formato' }}
```

El format de data accepta entre altres els següents camps:

- `yyyy`: El any amb 4 dígits. 2012, 1998
- `MM`: El mes en format numèric amb dos dígits. 01, 09, 12
- `dd`: El dia del mes amb dos dígits. 01, 21, 31
- `HH`: La hora en 24 Hores amb dos dígits. 01, 09, 11, 23
- `mm`: Els minuts amb dos dígits. 01, 09, 45, 59
- `ss`: Els segons amb dos dígits. 01, 09, 45, 59

La referència completa des formats de data la podem trobar en date.

Veiem ara un exemple:

```
<div>{{miFecha | date:'yyyy/MM/dd'}}</div>
```

Al mostrar la variable fecha es veurà primer el any, després els mesos i finalment els dies.
AngularJS suporta que en comptes de especificar directament el format de la data s'utilitzen els següents formats redefinits :

- `medium`
- `short`
- `fullDate`
- `longDate`
- `mediumDate`
- `shortDate`
- `mediumTime`
- `shortTime`

El significat exacte d'aquests formats depèn del idioma que s'hagui escollit, encara que es poden intuir per el nombre que tenen. Així que en la majoria de las ocasions s'haurien d'utilitzar aquests formats predefinits.

```
1    <div>{{miFecha | date:'shortDate'}}</div>
```

Ens hem de fixar en que al posar opcions del filtre que siguin un String s'ha de posar entre cometes.

currency

Aquest filtre mostra un nombre amb el símbol de la moneda local i amb el nombre de decimals correctes.

format:

```
{{ valor | currency }}
```

Veiem ara un exemple:

```
    <div>{{ importe | currency }}</div>
```

La variable importe es mostrarà per defecte amb el símbol del $ i amb 2 decimals però si estem en el idioma de "es-es" es mostrarà amb el símbol del "€" i també 2 decimals.
Aquest filtre també suporta passar-li directament el símbol de la moneda. Encara que el nº de decimals, separador, etc. seguiran sent el del idioma definit.

```
1    <div>{{ 12.45 | currency:'EUR' }}</div>
```

Mostrarà "12,45 EUR" en comptes de "12,45 €".

lowercase

Aquest filtre transformarà un String a minúscules.

format:

```
{{ valor | lowercase }}
```

Veiem ara un exemple:

```
    <div>{{ nombre | lowercase }}</div>
```

La variable `nombre` es mostrarà en minúscules.

uppercase

Aquest filtre transformarà un String a majúscules.

format:

```
{{ valor | uppercase }}
```

Veiem ara un exemple:

```
    <div>{{ nombre | uppercase }}</div>
```

La variable `nombre` es mostrarà en majúscules.

Internacionalització

Com hem vist, els filtres number , date i currency varien el seu format en funció del idioma i país. Com podem indicar al AngularJS el idioma i el país que volem utilitzar? Simplement tenim que cargar el fitxer JavaScript amb el idioma i país corresponent.
Quan descarreguem AngularJS, en la carpeta i18n tindrem tots els fitxers per a cada idioma/país. Així si per exemple volem utilitzar el idioma espanyol en Espanya només s'ha de carregar el fitxer:
i18n/angular-locale_es-es.js

Per tant en una aplicació en Espanya com el idioma espanyol només s' ha de incloure la següent línia després de carregar AngularJS:

```
<script src="//code.angularjs.org/1.2.19/i18n/angular-locale_es-
es.js"></script>
```

El que no suporta (per ara) AngularJS són més tipus de internacionalització com poden ser els misssatges, però hi han varis projectes en github amb filtres dedicats a això.

Recorda cargar el Script amb el idioma adequat després de carregar AngularJS:

```
<script
src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.min.js"></script>

<script src="//code.angularjs.org/1.2.19/i18n/angular-locale_es-es.js"></script>
```

Exemple

En el següent exemple s'apliquen els diferents filtres , mostrant totes les variants de formats predefinits de data . Per els exemples s'ha utilitzat el idioma espanyol en Espanya.

Veure exercici1.html i exercici1.js.

REFERÈNCIES

- / API Reference / ng / filter / number
- / API Reference / ng / filter / date
- / API Reference / ng / filter / currency
- / API Reference / ng / filter / lowercase
- / API Reference / ng / filter / uppercase
- / Developer Guide / i18n and l10n

Llistes

Aquests filtres s'apliquen per a modificar la forma en la que es mostren arrays de dades.

- orderBy: Permet ordenar un array
- limitTo: Limita el nombre d'elements d'un array
- filter: Filtra les dades del array.

El normal es utilitzar aquests filtres dins de la directiva `ng-repeat` per a modificar la manera en que es mostren els elements d'una taula.

orderBy

Permet ordenar un array de dades. El resultat serà el mateix array però ordenat.

format:
`{{array | ort`
Sent el nom de la propietat del objecte de cada element del array per la que volem ordenar. Si volem ordenar per més d'una columna posarem un array de Strings
`{{array | orderBy:['propiedad1','propiedad2',...,'propiedadn'] }}`
Veiem ara un exemple:

Suposem les següents dades

```
1       $scope.provincias = [{
2         codProvincia: "02",
3         nombre: "Albacete"
4       }, {
5         codProvincia: "03",
6         nombre: "Alicante/Alacant"
7       }, {
8         codProvincia: "04",
9         nombre: "Almería"
10      }];
```

Podem mostrar les províncies en una taula ordenades per la columna `nombre` amb la següent expressió:

```
1   <tr ng-repeat="provincia in provincias | orderBy:'nombre'">
2     <td>{{provincia.codProvincia}}</td>
3     <td>{{provincia.nombre}}</td>
4   </tr>
```

Si volguéssim ordenar per `nombre` i en cas de que dos es diguin igual [1]per `codProvincia` utilitzaríem un array amb els 2 Strings:

```
1   <tr ng-repeat="provincia in provincias | orderBy:['nombre','codProvincia']">
2     <td>{{provincia.codProvincia}}</td>
3     <td>{{provincia.nombre}}</td>
4   </tr>
```

Si volguéssim ordenar en comptes de forma ascendent de forma descendent no només caldria incloure el signe ”-” menys davant del nom de la columna:

```
1  <tr ng-repeat="provincia in provincias | orderBy:'-nombre'">
2    <td>{{provincia.codProvincia}}</td>
3    <td>{{provincia.nombre}}</td>
4  </tr>
```

AngularJS també permet crear la teva pròpia funció de ordenació però veurem res sobre ella ja que és complexa d’utilitzar i és poc útil.
Un problema d’aquest filtre és que al ordenar no té en compte els accents o coses similars, per lo que per exemple les lletres amb accents aniran després de totes les lletres *normals*.
En el exemple que hi ha a continuació la última província que es mostra és “Badajoz” i no apareix “Ávila”

limitTo

Aquest filtre permet limitar el nº d’elements que es mostren del array.

format:

```
{{array | limitTo:nombre }}
```

Essent “nombre” el nº de elements que volem mostrar del array.

Veiem ara un exemple:

Suposem les següents dades

```
1        $scope.provincias = [{
2          codProvincia: "02",
3          nombre: "Albacete"
4        }, {
5          codProvincia: "03",
6          nombre: "Alicante/Alacant"
7        }, {
8          codProvincia: "04",
9          nombre: "Almería"
10       }];
```

Podem mostrar només les 2 primeres províncies com:

```
<tr ng-repeat="provincia in provincias | limitTo:2">
  <td>{{provincia.codProvincia}}</td>
  <td>{{provincia.nombre}}</td>
</tr>
```

Si el valor es negatiu el que mostra és els 'n' últims elements del array en comptes dels 'n' primers.

Podem mostrar només les 2 últimes províncies com:

?

```
<tr ng-repeat="provincia in provincias | limitTo:-2">
  <td>{{provincia.codProvincia}}</td>
  <td>{{provincia.nombre}}</td>
</tr>
```

Recorda que `limitTo` permet nombres negatius per indicar que es mostrin els 'n' últims elements del array.
Encara està en la documentació oficial d'AngularJS, el filtre limitTo també es pot aplicar a Strings. En aquest cas el que fa és mostrar els "n" primers/últims caràcters del String.

filter

Ja que el filtre filter disposa de tantes opcions, li hem dedicat el següent tema complet a ell.

Consultar ejercici2.html i ejercici2.js.

REFERENCIES

- / API Reference / ng / filter / orderBy
- / API Reference / ng / filter / limiTo

Filtre filter

El filtre `filter` , el nom és poc afortunat, filtra el array per a que el array resultant només tingui certes dades del array original. Aquest filtre suporta diversos tipus de paràmetres així que anem a veure'ls un per un.
En tots els exemples anem a suposar les següents dades:

```
$scope.provincias = [{
    idProvincia: 1,
    nombre: "Palencia",
    comunidadAutonoma:"Castilla-Leon",
    idiomasCooficiales:false
```

```
6       }, {
7           idProvincia: 2,
8           nombre: "Castellón",
9           comunidadAutonoma:"Valencia",
10          idiomasCooficiales:true
11      }, {
12          idProvincia: 3,
13          nombre: "Alicante",
14          comunidadAutonoma:"Valencia",
15          idiomasCooficiales:true
16      }];
```

Recerca en totes les propietats

Anem a veure com buscar un text que estigui en qualsevol de les propietats dels objectes del array. Però no fa falta que el text i la propietat coincideixin exactament sinó només una part. Fent un símil com SQL és com si es fes un ” propiedad LIKE '%texto%' ” i utilitzant totes les propietats de cada objecte del array.

format:

```
{{ valor | filter:texto }}
```

Indiquem el valor texto amb un String amb el text a buscar, en **qualsevol** propietat dels objectes d’array.
Si apliquem el filtre de la següent forma:

```
1   <tr ng-repeat="provincia in provincias | filter:'encia'">
2     <td>{{provincia.codProvincia}}</td>
3     <td>{{provincia.nombre}}</td>
4   </tr>
```

Mostrarà les 3 províncies ja que el text “encia” està en la propietat nom de “Palencia” i en la propietat “comunidadAutonoma” de “Alicante” i “Castellón”.

Recerca en una propietat

Podem restringir per a que es busqui en només una propietat de la següent forma:

format:

```
{{ valor | filter:{ propiedad:texto } }}
```

Essent texto un String amb el text a buscar en la propietat anomenada `propiedad`

```
1  <tr ng-repeat="provincia in provincias | filter:{
   nombre:'encia'}'">
2    <td>{{provincia.codProvincia}}</td>
3    <td>{{provincia.nombre}}</td>
4  </tr>
```

Ara només busca el text "encia" en la propietat `nombre`.

Recerca en varies propietats

Si volem buscar en més d'una propietat i fent un **AND** entre elles podem utilitzar:

format:

```
{{ valor | filter:{ propiedad1:texto1, propiedad2:texto2 } }}
```

```
1  <tr ng-repeat="provincia in provincias | filter:{
   nombre:'encia',comunidadAutonoma:'illa'}'">
2    <td>{{provincia.codProvincia}}</td>
3    <td>{{provincia.nombre}}</td>
4  </tr>
```

Ara només buscarà aquelles províncies que el nom contingui el text "encia" **i** que la comunitat autònoma contingui el text "illa".

Recerca combinada

Utilitzant el nom de propietat "$" es pot buscar un text en totes les propietats però a més afeguir una altre condició com un **AND** com acabem de fer. Es a dir que la propietat "$" és com si fos el nom de qualsevol propietat.
Veiem un exemple:

```
1  <tr ng-repeat="provincia in provincias | filter:{
   $:'encia',idiomasCooficiales:true}'">
2    <td>{{provincia.codProvincia}}</td>
3    <td>{{provincia.nombre}}</td>
4  </tr>
```

Ara buscarem el text "encia" en qualsevol propietat però a més la propietat idiomasCooficiales té que ser true.
Veiem que és una avantatge a utilitzar

```
array | filter:{$:texto}
```

Amb respecte a simplement utilitzar

```
array | filter:texto
```

Ja que en el primer cas podem afegir condicions **AND** amb el format

```
array | filter:{$:texto,propiedad:valor}
```

mentres que en el segon cas no es pot.

Comparador

Hem vist que el filtre filter comprova que el que busquem estigui contingut dins de la propietat en la que busquem, però, i si volem que només ho trobi si és la paraula sencera? I si volem que trobi una paraula només si comença per el text per el que es filtra?, etc.
Per ajudar-nos en tot això, el filtre filter disposa d'un segon paràmetre que permet indicar com es fa la comparació entre els textos, veiem els 3 possibles valors.

false

Posar false és la funcionalitat per defecte, per lo tant és lo mateix que no posar res. I com ja hem dit, buscarà que el text a buscar estigui contingut en la propietat. És com en SQL `propiedad LIKE "%textoBuscar%"`.

```
1 <tr ng-repeat="provincia in provincias | filter:{nombre:'encia'}:false">
2   <td>{{provincia.codProvincia}}</td>
3   <td>{{provincia.nombre}}</td>
4 </tr>
```

Busca en la propietat "nombre" si **conté** la paraula "encia".

true

Si es posa el valor true, en aquest cas el text a buscar té que ser igual al valor de la propietat. És com en SQL propiedad = textoBuscar

```
1 <tr ng-repeat="provincia in provincias | filter:{nombre:'encia'}:true">
2   <td>{{provincia.codProvincia}}</td>
3   <td>{{provincia.nombre}}</td>
4 </tr>
```

Recerca en la propietat "nombre" si el seu valor és **exactament** la paraula "encia".

funcion

En aquest cas no posem un valor booleà com true o false sinó el nom d'una funció creada per nosaltres en el $scope i a la que se li pasarà 2 arguments. El primer argument és el valor de la propietat i el segon argument és el text a buscar. Si la funció retorna true és que són iguals i si retorna false és que són diferents. En el nostre exemple anem a suposar que tenim la següent funció en el $scope

```
1  $scope.comparator = function(actual, expected) {
2      if (actual.indexOf(expected)===0) {
3        return true;
4      } else {
5        return false;
6      }
7  };
```

Aquesta funció retorna true si el valor de actual comença per el valor de `expected`

```
1  <tr ng-repeat="provincia in provincias | filter:{
   nombre:'encia'}:comparator">
2    <td>{{provincia.codProvincia}}</td>
3    <td>{{provincia.nombre}}</td>
4  </tr>
```

Busca en la propietat "nombre" si el seu valor **comença** amb la paraula "encia".Per fer-ho s'ha de cridar a la funció comparator que com ja hem explicat només retorna true si el valor comença per la paraula buscada. Veiem una taula comparant més característiques dels 3 modes:

Valor paràmetre	El texto a buscar es ""	Sensible a majúscules/minúscules	Sensible a accents, dièresis, etc.
false	Se retornen totes las files	NO	tenen que coincidir els accents, dièresis, etc.
true	No se retorna ninguna fila	SI	Tenen que coincidir els accents, dièresis, etc.
Funció JavaScript	Personalitzat segons el que faci la funció	Personalitzat segons el que faci la funció	Personalitzat segons el que faci la funció

Com veiem al utilitzar una funció ens permet personalitzar tot el que vulguem, com comparar les dades, encara que realment no permet canviar la consulta de recerca sinó només la forma de comparar-les 2.

internacionalització

Encara segur que podríem fer-ho, aquí hi ha un exemple de funció de comparació per a Strings que és insensible a majúscules / minúscules però que també troba els elements independentment dels accents, dièresi i circumflexos:

```
1   /**
2    * Esta función cambia todas las vocales con acentos, diéresis y
    circunflejos
3    * por la vocal sin ellos y también la transforma a mayúsculas.
4    */
5   function normalize(texto) {
6     texto = texto.replace(/[áàäâ]/g, "a");
7     texto = texto.replace(/[éèëê]/g, "e");
8     texto = texto.replace(/[íìïî]/g, "i");
9     texto = texto.replace(/[óòôö]/g, "o");
10    texto = texto.replace(/[úùüü]/g, "u");
11    texto = texto.toUpperCase();
12    return texto;
13  }
14
15  /** Esta función se puede usar como comparator en el filter **/
16  $scope.comparator = function(actual, expected) {
17      if (normalize(actual).indexOf(normalize(expected))>=0) {
18        return true;
19      } else {
20        return false;
21      }
22  };
```

- Línia 5: Aquesta funció transforma qualsevol text traient a qualsevol vocal els accents, dièresi i circumflexos.

- Línia 16: Aquesta es la funció de comparació que utilitzarem en filter. La seva entrada són els 2 texts a comparar.
- Línia 17: Aquesta línia transforma els 2 texts utilitzant la funció normalize que treu els accents, dièresi i circumflexos. Llavors al estar lliure d'ells, simplement els compara utilitzant la funció indexOf.

Ara podríem utilitzar la nostra funció comparator en filter i trobaria qualsevol paraula encara que escrivim malament els accents, dièresi o circumflexos.

Recerca personalitzada

Fins ara hem vist com fer filtres mirant en varies propietats però sempre era fent un **AND** entre elles. AngularJS permet que li passem una funció per que decidim por nosaltres mateixos com filtrar les dades permetent fer filtres tot el complexes que vulguem. En aquest cas al filtre filter se li passa un únic paràmetre amb el nom d'una funció creada per nosaltres en el $scope . Aquesta funció accepta un únic paràmetre amb un element d'array i tenim que retornar true en cas de que vulguem que aquest element es mostri false si no volem que es mostri.
Veiem un exemple. Suposem que en el $scope tenim la següent funció:

```
1  $scope.consultaPersonalizada=function(value) {
2    if ((value.idiomasCooficiales===true) || (value.idProvincia>2))
   {
3      return true;
4    } else {
5      return false;
6    }
7  }
```

És una funció a la que li passarem un valor del array i ens retorna true si value.idiomasCooficiales===true **OR** value.idProvincia>2. Observa com hem utilitzat el operador OR que fins ara no havíem pogut utilitzar.
Ara podem simplement utilitzar aquesta funció en el filtre filter.

```
1  <tr ng-repeat="provincia in provincias |
   filter:consultaPersonalizada">
2    <td>{{provincia.codProvincia}}</td>
3    <td>{{provincia.nombre}}</td>
4  </tr>
```

En aquest cas només es mostrarà aquelles províncies que tinguin un idioma cooficial i que `idProvincia` sigui major que 2

En cas de utilitzar la funció personalitzada de filtro ja que no es pot passar un segon argument amb la funció de `comparator` ja que no té sentit doncs la funció personalitzada de filtre s'encarrega de fer tot el treball.

Exemple

En aquest exemple podem escriure en la caixa de text i podrem veure 3 exemples de com es comparen les dades en el filtre com:

- `false`
- `true`
- funció personalitzada de comparació que només retorna `true` si el text a buscar està al principi del valor de la propietat però que es sensible a majúscules / minúscules.

Provem de escriure:

- ca
- Ca
- llón
- llon
- Castellón
- castellon

I observa com funciona cada una de les 3 formes.

Per últim hi ha una última taula en la que es veu la funció personalitzada.

Consultar exercici3.html i exercici3.js.

Per obtenir els CDNS de les normalitzacions lingüístiques de cada idioma i país. https://cdnjs.com/libraries/angular-i18n

Formularis

Introducció a els formularis

AngularJS permet saber la següent informació d'un formulari HTML:

- Si s'ha modificat o no.
- Si son vàlids els seus camps o no.

En aquesta tema anem a posar exemple seguint el següent formulari HTML:

```
1   <form name="miFormulario" >
2     Nombre:<input type="text" ng-model="model.nombre" name="nombre" required >
```

```
3     <br>
4     Correo eletronico:<input type="text" ng-model="model.email" name="email" required>
5   </form>
```

Aquest formulari conté 2 validacions:

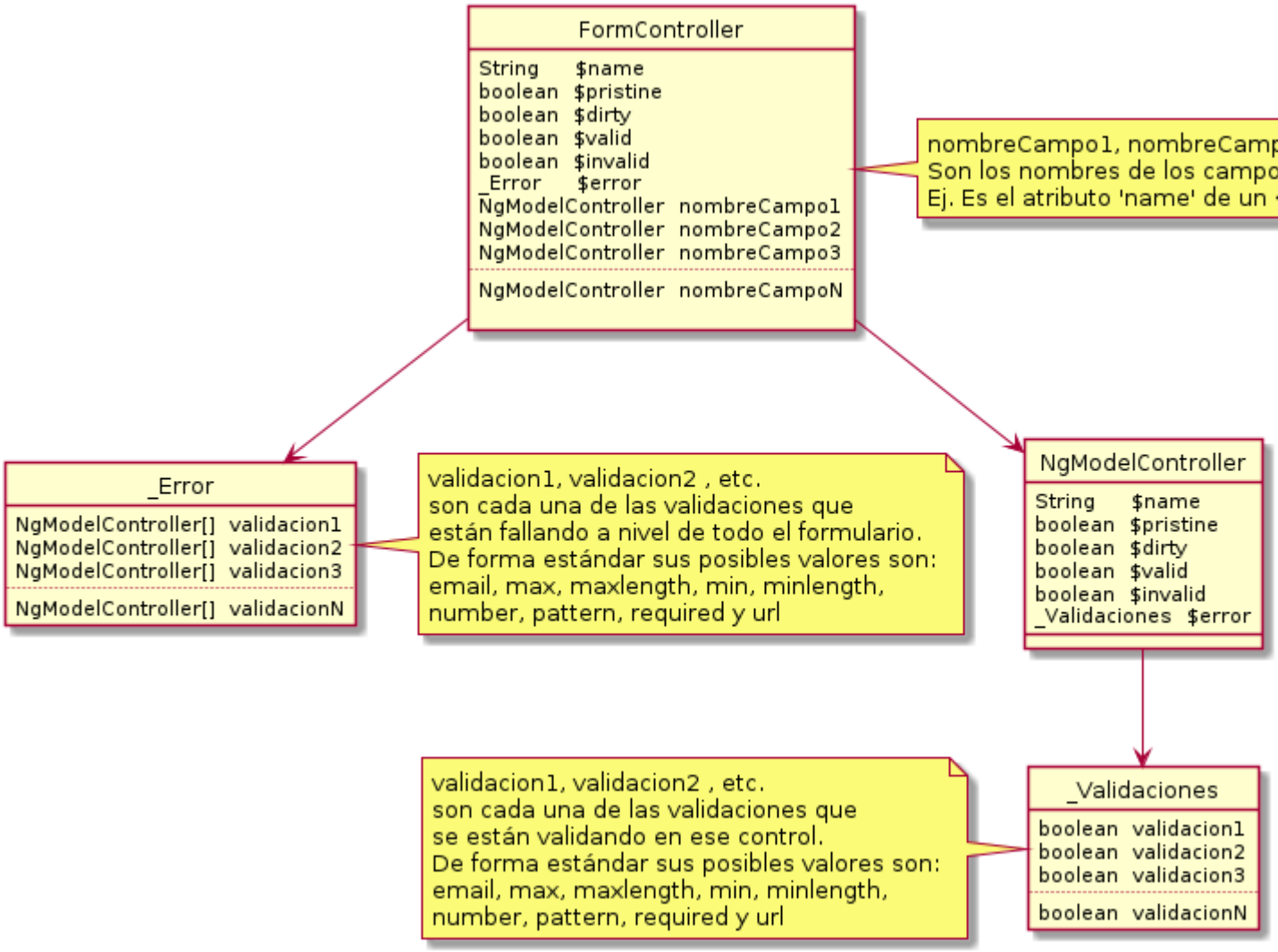
- El camp “nombre” és requerit
- El camp “email” es requerit

FormController

Per accedir a la informació que proporciona AngularJS sobre el formulari HTML tenim la classe FormController. A la instància d’aquesta classe s’accedeix a través del `$scope`de el nostre controlador. El `$scope` té una propietat que es diu com el nom[1)] del formulari HTML i que és una instància de la classe FormController
La forma de fer-ho en el exemple seria la següent:

```
?
1   var formController=$scope.miFormulario
```

No es possible accedir a la propietat del formulari fins que no s’hagi acabat la inicialització del controlador. És a dir que fins que no s’ha acabat de executar la funció del controlador no existeix la propietat.
Veiem ara un diagrama UML ón s’expliquen las diferents propietats de la classe FormController



- Tant com la classe FormController com la classe NgModelController disposen de més propietats i mètodes, alguns dels quals es veurà en altres unitats i alguns altres no els veurem en aquest curs.
- La classe NgModelController no es veurà en profunditat ja que és necessari tenir coneixements avançats de directives per a comprendre completament.
- Durant aquesta unitat anem explicant cada una de les propietats d'aquestes classes.
- Els noms de classes de `_Error` y `_Validaciones` no són noms que existeixen com a tal en AngularJS però els hem creat per a representar les formes de les propietats `FormController.$error` y `NgModelController.$error`. És a dir que tenen aquesta informació però en cap lloc de la documentació d'AngularJS apareixen aquests 2 noms.

Que anem a aconseguir amb aquest petit monstre de classe enrevessat? Com ja hem dit, per poder saber en tot moment el estat del formulari i de tots els seus camps.

Canvis

El primer que podem saber del formulari és si algo s'ha modificat en els camps del formulari, per el que AngularJS disposa de les propietats:

- `$pristine`: Val `true` si el formulari o camp encara no ha sigut modificad per el usuari, si no val `false`.
- `$dirty`: Val `true` si el formulari o camp ja ha sigut modificat per el usuari, si no val `false`.

Com podem imaginar `$pristine` sempre val el contrari de `$dirty` i viceversa.
Aquestes propietats es poden aplicar tant a nivell de formulari com a nivell d'un camp concret.

Veiem ara alguns exemples d'ells:

Expresión JavaScript	Significado
`$scope.miFormulario.$pristine`	Valdrà `true` si el usuari no ha modificad cap camp del formulari "miFormulario"
`$scope.miFormulario.$dirty`	Valdrà `true` si el usuari ja ha modificat algun camp del formulari "miFormulario"
`$scope.miFormulario.nombre.$pristine`	Valdrà `true` si el usuari no ha modificat el camp "nombre" del formulari "miFormulario"
`$scope.miFormulario.nombre.$dirty`	Valdrà `true` si el usuari ja ha modificat el camp "nombre" del formulari "miFormulario"
`$scope.miFormulario.email.$pristine`	Valdrà `true` si el usuari no ha modificat el camp "email" del formulari "miFormulario"
`$scope.miFormulario.email.$dirty`	Valdrà `true` si el usuari ja ha modificat el campo "email" del formulari "miFormulario"

Validesa

El següent que podem comprovar és si els valors del formulari o d'algun camp són o no vàlids [2]. Per això AngularJS disposa de les següents propietats:

- `$valid` : Vae `true` si el formulari o el camp son vàlids, és a dir, si compleixen totes les validacions que s'han indicat en els camps.
- `$invalid` : Vale `true` si el formulari o el camp són invàlids, és a dir, si s'incompleix alguna de les validacions que s'han indicat en los camps.

Com podem imaginar `$valid` sempre val lo contrari de `$invalid` i viceversa.
Aquestes propietats es poden aplicar tant a nivell de formulari com a nivell d'un camp concret.

Veiem ara alguns exemples d'ells:

Expresión JavaScript	Significado
`$scope.miFormulario.$valid`	Valdrà `true` si totes les validacions de tots els camps del formulari "miFormulario" son vàlides
`$scope.miFormulario.$invalid`	Valdrà `true` si alguna validació d'algun dels camps del formulari "miFormulario" es invàlida

`$scope.miFormulario.nombre.$valid`	Valdrà `true` si totes les validacions del camp "nombre" del formulari "miFormulario" són vàlides
`$scope.miFormulario.nombre.$invalid`	Valdrà `true` si alguna validació del camp "nombre" del formulari "miFormulario" es invàlida
`$scope.miFormulario.email.$valid`	Valdrà `true` si totes les validacions del camp "email" del formulari "miFormulario" son vàlides
`$scope.miFormulario.email.$invalid`	Valdrà `true` si alguna validació del camp "email" del formulari "miFormulario" es invàlida

Exemple

El exemple conté el següent formulari:

```
1  <form name="miFormulario" novalidate>
2    Nombre:<input type="text" ng-model="model.nombre" name="nombre" required >
3    <br>
4    Correo electrónico:<input type="text" ng-model="model.email" name="email" required>
5  </form>
```

En el que els camps "nombre" i "email" són requerits. Es pot veure el estat de totes les variables `$pristine`, `$dirty`, `$valid` i `$invalid` a mesura que es canvien les dades del formulari.

Veure ejercici4.html i ejercici4.js.

Validació de formularis

En aquest tema anem a veure les validacions estàndard que té angular deixant per a més endevant la creació de noves validacions.

Tipus de Validacions

AngularJs disposa de 9 tipus de validacions diferents:

- `email`: El camp té que tenir el format d'un correu electrònic
- `max`: El camp té que tenir un valor màxim
- `maxlength`: El camp té que tenir un nº màxim de caràcters
- `min`: El camp té que tenir un valor mínim
- `minlength`: El camp té que tenir un nº mínim de caràcters
- `number`: El camp té que ser un nombre
- `pattern`: El camp té que seguir una expressió regular
- `required`: el camp es requerit

- `url`: El camp té que tenir el format de una URL

Per a que totes aquestes validacions funcionin és necessari que tant el `<form>`como el `<input>`, `<select>` , etc. tinguin la propietat `name`.

```
<form name="nombre">

<input name="nombre">

<select name="nombre">
</form>
```

Passem ara a veure com s'apliquen cada una d'elles.

email

Haurem d'indicar en el `<input>` que `type="email"`.

```
<input type="email" ng-model="model.correo" name="correo" >
```

max

Només s'aplica a `<input>` on el `type="number"`. Haurem de indicar la directiva `max="valor"`, sent `valor` el valor màxim que pot tenir.

```
<input type="number" ng-model="model.edad" name="edat" max="99" >
```

maxlength

Només s'aplica a `<input>` cuyo `type="number"` o `type="text"` o `<textarea>`. Haurem d'indicar la directiva `ng-maxlength="valor"`, sent `valor` la mida màxim que pot tenir en caràcters.

```
<input type="text" ng-model="model.nombre" name="nombre" ng-maxlength="50" >
```

min

Només s'aplica a `<input>` cuyo `type="number"`.
Haurem d'indicar la directiva `min="valor"`, sent `valor` el valor mínim que pot tenir.
?

```
<input type="number" ng-model="model.edad" name="edad" min="18" >
```

minlength

Només s'aplica a `<input>` ón `type="number"` o `type="text"` o `<textarea>`. Haurem d'indicar la directiva `ng-minlength="valor"`, sent `valor` la mida mínim que pot tenir en caràcters.

```
1    <input type="text" ng-model="model.nombre" name="nombre" ng-minlength="3" >
```

number

Haurem d'indicar en el `<input>` que `type="number"`.

```
1    <input type="number" ng-model="model.edad" name="edad" >
```

pattern

Només s'aplica a `<input>` sempre que sigui `type="number"`, `type="text"` o `type="email"` o `<textarea>`. Haurem d'indicar la directiva `ng-pattern="valor"`, sent `valor` la expressió regular que té que complir el valor.

```
1    <input type="text" ng-model="model.nombre" name="nombre" ng-pattern="/^[a-zA-Z]*$/" >
```

Recorda que:

- S'ha de posar les barres al principi i al final
- Si volem que la expressió s'apliqui a tot el valor posa "^" al principi i "$" al final

Aquesta directiva permet que la expressió regular aquest en una propietat del `$scope`. En aquest cas té que posar-se així:

```
1    <input type="text" ng-model="model.nombre" name="nombre" ng-pattern="patternNombre" >
```

I en `$scope` indicar el següent:

```
1    $scope.patternNombre=/^[a-zA-Z]*$/;
```

required

S'aplica a qualsevol `<input>` , `<select>` o `<textarea>`.
Té que posar la directiva `required` o `ng-required`.
La diferència entre ells és que al posar `required` el valor es requerit però al posar `ng-required` se li podria posar un valor al atribut `ng-required` que fos `true` o `false`pero que aquest valor vingui del `$scope`.

```
1    <input type="text" ng-model="model.nombre" name="nombre" required >
```

```
1    <input type="text" ng-model="model.nombre" name="nombre" ng-required="requeridoNombre" >
```

En aquest segon exemple per a que fos requerit `model.nombre`, hauríem de tenir en el `$scope` :

```
1    $scope.requeridoNombre=true
```

I per que **no** fos requerit `model.nombre`, hauríem de tenir en el `$scope` :

```
1    $scope.requeridoNombre=false
```

url

Hauríem d'indicar en el `<input>` que `type="url"`.

```
1    <input type="url" ng-model="model.sitioweb" name="sitioweb" >
```

Comprovant les validacions

Ara que ja sabem com posar les validacions falta saber si una validació ha fallat. Hi ha dos formes de saber-ho:

- Sobre un camp
- Sobre un tipus de validació

Sobre un camp

Si sabem un camp, podrem esbrinar les validacions que ha fallat. Per explicar-ho anem a fer-ho amb uns exemples:

Exemple 1

Si volem saber si ha fallat la validació `required` en el campo `nombre` del formulari `miFormulario` haurem de veure el valor de la següent expressió:

```
1    $scope.miFormulario.nombre.$error.required
```

- Si val `true` és que la validació ha fallat i no està el valor
- Si val `false` es que la validació és correcta i té valor
- Si val `undefined` és que no s'està validant la validació `required`

Exemple 2

Si volem saber si ha fallat la validació `email` en el camp `correo` del formulari `miFormulario` hauríem de veure el valor de la següent expressió:

```
1    $scope.miFormulario.correo.$error.email
```

- Si val `true` és que la validació ha fallat i el valor no és un correu electrònic
- Si val `false` és que la validació és correcta i el valor és un correu electrònic

- Si val `undefined` és que no s'està validant la validació de `email`

El que ens retorna la expressió `$scope.miFormulario.nombre` o `$scope.miFormulario.correo` són objectes de tipus NgModelController sobre els que podrem accedir a totes les seves propietats.
Gràcies a que JavaScript permet saber que propietats hi ha en un objecte, podríem recórrer totes les propietats per tenir una llista de tot el que s'està fallant.

Sobre un tipus de validació

Donat un tipus de validació podem saber quants camps no compleixen aquesta validació. Per explicar-ho anem a fer-ho amb uns exemples:

Exemple 1

Si volem saber quants camps han fallat la validació de `required` del formulari `miFormulario` haurem de veure el valor de la següent expressió:

```
1    $scope.miFormulario.$error.required
```

El resultat és aquesta expressió

- Si val `false` és que no hi ha cap camp que incumpleix aquesta validació.
- Si val un array. El contingut del array són objectes de la classe NgModelController corresponents a cada un dels camps en els que ha fallat aquesta validació
- Si val `undefined` és que cap camp está validant aquesta validació. En el nostre exemple és que ningú ha posat `required` en cap camp.

Exemple 2

Si volem saber quants camps han fallat la validació de `email` del formulari `miFormulario` haurem de veure el valor de la següent expressió:

```
1    $scope.miFormulario.$error.email
```

El resultat és aquesta expressió

- Si val `false` és que no hi ha cap camp que incompleixi aquesta validació.
- Si val un array. El contingut del array son objectes de la classe NgModelController corresponents a cada un dels camps en els que ha fallat aquesta validació.
- Si val `undefined` és que cap camp està validant aquesta validació. En el nostre exemple significaria que no hi ha camps del tipus `type="email"`.

Gràcies a que JavaScript permet saber que propietats hi ha en un objecte, podríem recórrer totes les propietats per a tenir una llista de tot el que està fallant.

novalidate

El últim que ens queda per explicar és el atribut `novalidate` que hauríem de posar en el tag `<form>`. Aquest atribut es posa per evitar que el propi navegador es posi a fer les seves pròpies validacions amb els seus propis missatges o estils CSS i que xoquin amb les que està fent AngularJS.

```
1    <form name="miFormulario" novalidate >

2    </form>
```

Recorda posar el atribut `novalidate` en el tag `<form>`.

Consultar Ejercici5.html i Ejercici5.js.