AngularJS Training

Día 1

Basado en AngularJS 1.4.6

Cristhian Amaya

cristhian@lecoati.com

@_camaya

Agenda

- Introducción
- Módulos
- Rutas
- Controladores
- Vistas
- Servicios
- Directivas
- Filtros

Antes de empezar

- https://tlk.io/angulartraining
- npm install -g http-server (https://github.com/indexzero/http-server)
- git clone https://github.com/Lecoati/Angular-Training-New-Home.git
- Desactivar cache en las herramientas de desarrollador
- ¡Demo!

Introducción

- ¿Qué es una SPA?
- ¿Qué es AngularJS?
- ¿Por qué Angular?
- Hola Mundo
- Visión Global
- Estructura de Ficheros

¿Qué es una SPA?

- Aplicaciones de una sola página
- Son más rápidas
- Se asemejan más a una aplicación de escritorio

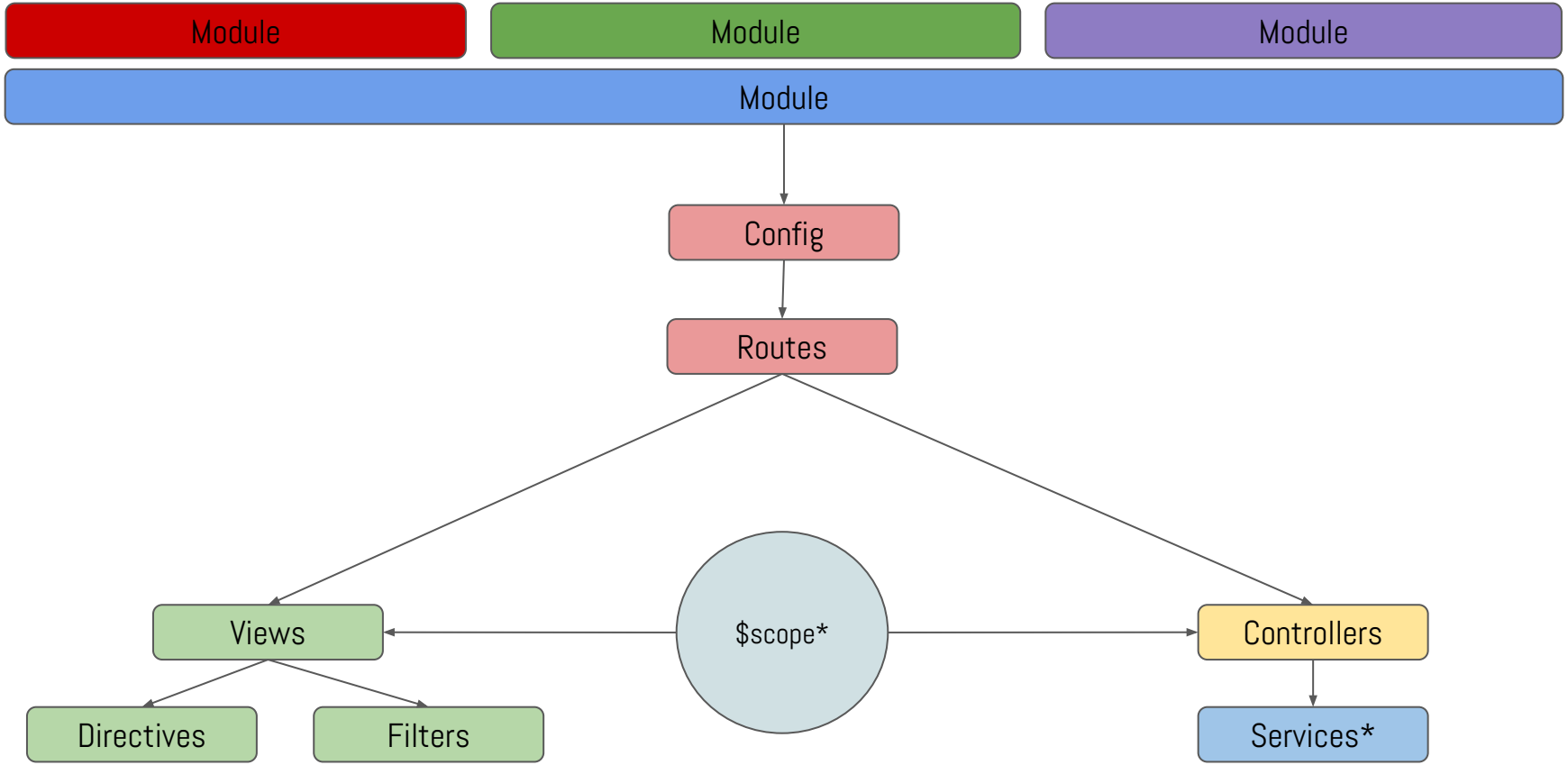
¿Qué es AngularJS?

- Framework MV*
- Ofrece todas las herramientas necesarias para construir una SPA
- Declarativo
- HTML para aplicaciones web
- Open Source, mantenido por Google

¿Porqué AngularJS?

- El framework nos da una estructura
- Es muy fácil de testear
- Reduce los tiempos de desarrollo
- Comunidad gigantezca
- https://www.madewithangular.com/

¡Hola Mundo!


Module
Module
Module
Module
Config
Routes
Views
$scope*
Controllers
Directives
Filters
Services*

Visión Global

- Las vistas y controladores son los componentes base
- Las rutas (cuando las tenemos), nos proveen el contexto para los controladores y las vistas
- Las directivas y los filtros extienden la vista
- Los servicios extienden los controladores

Estructura de ficheros

Por Tipo

```
~app-by-type/
 |+controllers/
 |+directives/
 |+filters/
 |+services/
 |+views/
 `-newhome.module.js
```

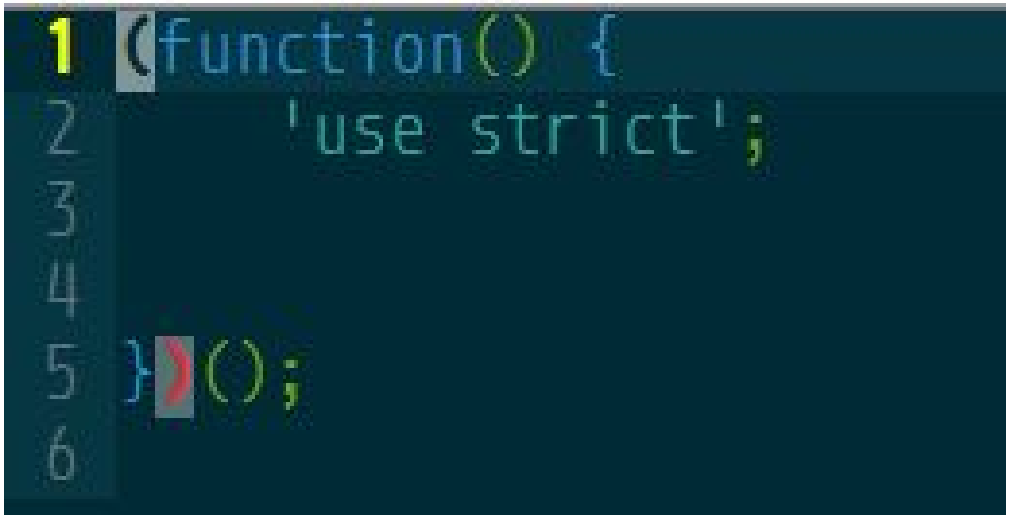
Por Feature

```
~app/
 |+animals/
 |+common/
 |+contact/
 |+core/
 |+data/
 |+home/
 |+layout/
 |+widgets/
 `-newhome.module.js
```

Estructura de ficheros

- Organiza las aplicaciones por feature
- Elige una convención
  - https://github.com/johnpapa/angular-styleguide
  - https://github.com/toddmotto/angularjs-styleguide
  - https://github.com/mgechev/angularjs-style-guide
  - https://github.com/ngbp/ngbp
  - ¡Crea la tuya!

IIFE (Immediately Invoked Function Expressions)



Módulos

- Contenedores
- Pueden depender de otros módulos
- Se pueden ver como namespaces
- Se componen de bloques de configuración y ejecución
- Solo se cargan una vez
- Reusabilidad

Tipos de módulos

- Módulos de Angular
- Módulos de terceros
- Módulos personalizados

Ejemplo 02

Ejercicio 01 - Crear módulo de la aplicación

1. **git checkout 01application-module**
2. Crear carpeta **app** en la raíz y carpeta **core** bajo **app**
3. Crear fichero **core.module.js** dentro de la carpeta **core** y definir módulo **newHome.core** sin dependencias.
4. Crear fichero **newhome.module.js** dentro de la carpeta **app** y definir módulo **newHome** con dependencia del módulo **newHome.core**
5. Agregar en index.html la referencia a los scripts que hemos creado, primero newHome y luego core
6. Activar módulo **newHome** en el **markup**
7. Abrir navegador y comprobar que no tenemos errores en la consola

Rutas

- Permiten navegar por las diferentes “páginas” de una aplicación
- Define controladores y vistas base
- Usaremos ngRoute

Ejemplo 03

Ejercicio 02 - Ruta homepage

1. **git checkout 02homepage-route --force**
2. Añadir referencia al script **angular-route** que está en la carpeta vendor. Tiene que ir después de angular
3. Añadir dependencia a **ngRoute** en módulo **core**
4. Crear fichero **home.module.js** dentro de la carpeta **home** y definir módulo **newHome.home** sin dependencias.
5. Crear fichero **home.routes.js** dentro de la carpeta **home** y añadir ruta "/"para la homepage con controlador **HomeController**, vista **home.html** (ya existen) y **controllerAs home**
6. Añadir **dependencia** del módulo **home** al módulo **newhome**
7. Actualizar index.html con las referencias a los scripts de la carpeta **home**
8. Abrir navegador. Deberíamos ver "Bienvenidos a New Home"

Controladores

- Proveen los datos que necesita una vista
- Deben contener únicamente la funcionalidad que requiere la vista
- No hay lógica de negocio
- No se debe manipular el DOM

Ejemplo 04

Ejemplo 05

$Scope Vs controller as

- $scope hereda prototipicamente de su $scope padre hasta $rootScope
- $scope no es solo para binding
  - $watch
  - $watchCollection
  - $broadcast
  - $emit
  - $on
- Controller as es más limpio
- Controller as nos provee un contexto

Ejercicio 03 - Controlador homepage

1. **git checkout 03homepage-controller --force**
2. Recordemos que el controlador y la vista están configurados en **home.routes.js**
3. Crear fichero **home.controller.js** dentro de la carpeta **home** y definir controlador **HomeController** usando la sintaxis **controller as**
4. Definir una propiedad llamada **title** con cualquier valor
5. Mostrar la propiedad **title** en la vista **home.html**
6. Abrir navegador. Deberíamos ver el valor de la propiedad que creamos en el controlador

Vistas

- Markup resultante después de que Angular compila el DOM
- Encapsula funcionalidades complejas declarativamente
- Se encarga de mostrar el modelo
- Hace de puente entre el usuario y nuestra aplicación

ng-repeat

Nos permite iterar sobre una colección de elementos y mostrar los elementos en la vista

Ejemplo 06

Ejercicio 04 - Lista de animales

1. **git checkout 04animals-list --force**
2. En el controlador hay definido un array de animales
3. Abrir **home.html** y usar **ng-repeat** para iterar sobre la lista de animales y mostrar los datos de cada uno.
4. Abrir el navegador. Deberíamos ver las tarjetas de 3 animales

Servicios

- Funcionalidades comunes
- Se consumen usando inyección de dependencias
- Son singletons
- Se inician perezosamente
- Aquí es donde queremos poner lógica

Ejemplo 07

Ejercicio 05 - Encapsulación de datos

1. **git checkout 05data-encapsulation --force**
2. Crear carpeta **data** bajo carpeta **app**
3. Crear fichero **data.module.js** dentro de la carpeta **data** y definir módulo **newHome.data** sin dependencias.
4. Crear fichero **animals.service.js** dentro de la carpeta **data** y definir servicio **animalsService**
5. Crear función **getAll** dentro del servicio **animalsService** que retorne los datos que hay ahora en el controlador **HomeController** y exponer dicha función
6. Agregar dependencia al módulo **newHome.data** en el módulo **newHome.core**
7. Inyectar el servicio **animalsService** en el controlador **HomeController**
8. Modificar controlador para obtener los datos usando la función **getAll** del servicio
9. Agregar referencias a los nuevos scripts en index.html
10. Abrir navegador. Deberíamos seguir viendo la misma lista de animales de antes.

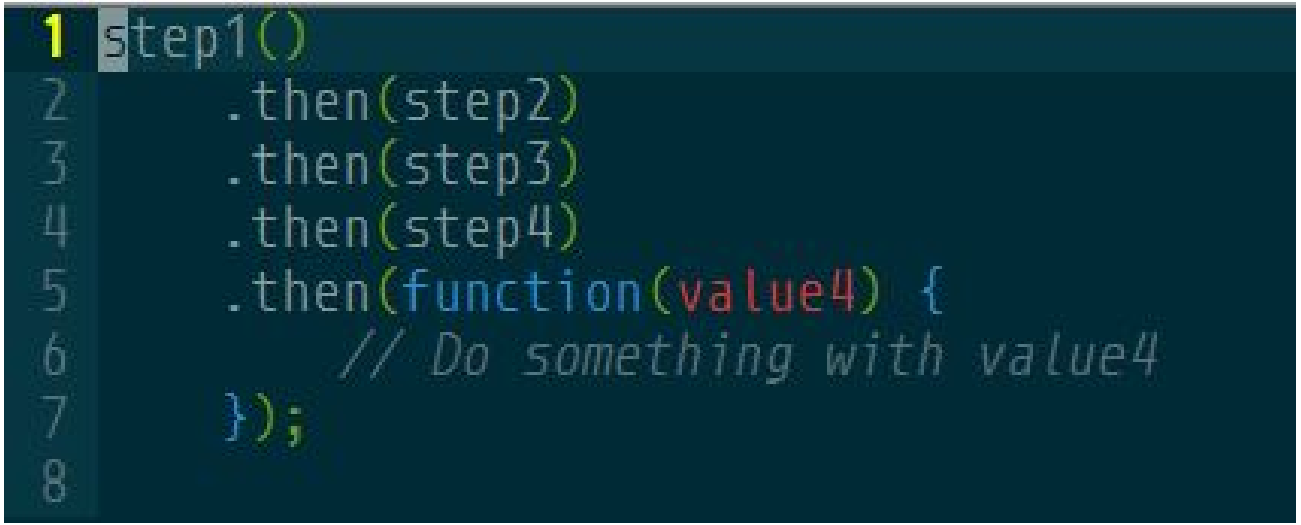
Promises

- Evolución de los callbacks
- Permiten hacer operaciones asíncronas
- Los métodos principales de una promise son: **then**, **catch** y **finally**
- Algunos servicios de Angular usan promises
- Podemos crear nuestras propias promises usando el servicio **$q**

Callback hell

```
step1(function(value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

Promises heaven



```
step1()
    .then(step2)
    .then(step3)
    .then(step4)
    .then(function(value4) {
        // Do something with value4
    });

```

Ejemplo 08

Ejercicio 06 - Uso de promises para leer datos

1. **git checkout 06using-promises --force**
2. **Inyectar** el servicio **$q** en el servicio **animalsService**
3. Modificar método **getAll** para que devuelva una **promise**
4. Modificar **HomeController** para obtener los datos una vez que la promise se resuelva
5. Abrir navegador. Deberíamos seguir viendo la misma lista de animales de antes.

El servicio $http

- Nos permite comunicarnos con un servidor mediante AJAX
- Contiene métodos que simulan los verbos REST (get, post, put, delete)
- Esta construido sobre promises
- Equivalente a $.ajax de jQuery

Ejemplo 09

Ejercicio 07 - Uso de $http para obtener datos

1. **git checkout 07using-$http --force**
2. Crear fichero **data.constants.js** bajo la carpeta **data** y definir una constante llamada **API_ENDPOINT** con el valor **app/data/animals.json**
3. Inyectar el servicio **$http** en el servicio **animalsService**
4. Inyectar la constante **API_ENDPOINT** en el servicio **animalsService**
5. Modificar método **getAll** para que devuelva una promise usando **$http** y la constante **API_ENDPOINT**
6. Abrir navegador. Deberíamos seguir viendo la misma lista de animales de antes.

ng-click

Nos permite especificar el comportamiento que queramos que tenga un elemento cuando se haga clic sobre el.

Ejemplo 10

Ejercicio 08 - Añadiendo likes

1. **git checkout 08adding-likes --force**
2. En el controlador **HomeController** añadir función **addLike(animal)**
3. Esta función tiene que incrementar en uno el número de likes del animal
4. En el **markup,** en el **icono del corazón**, usar ng-click para llamar a la función **addLike** que acabos de crear
5. Abrir navegador. Cuando hagamos click en el icono del corazón, se debería incrementar el número de likes.

Ejercicio 09 - Guardando likes en el servidor

1. **git checkout 09saving-likes**
2. En el fichero **data.contants.js** modificar valor de la constante **API_ENDPOINT** por: **http://angular-animals.azurewebsites.net/api/animals/**
3. El **endpoint** para añadir un like es: **PUT animals/1/likes** donde 1 es el id del animal.
4. Añadir función **addLike(animal)** dentro del servicio **animalsService** que haga una llamada al endpoint de likes usando el servicio **$http** y la constante **API_ENDPOINT** y exponer dicha función
5. Modificar **HomeController** para que use la función **addLike** del servicio **animalsService**
6. Incrementar el contador de likes cuando se resuelva la **promise** de **addLike**
7. Abrir el navegador. Cuando hagamos click sobre el icono del corazón, deberíamos seguir viendo como se incrementa el contador de likes. La diferencia es que ahora este valor está persistiendo.

Ejercicio 10 - Añadiendo página de detalle

1. **git checkout 10adding-animal-details --force**
2. Crear carpeta **animals** bajo **app**
3. Crear fichero **animals.module.js** dentro de la carpeta **animals** y definir módulo **newHome.animals** sin dependencias.
4. Crear fichero **animal.routes.js** dentro de la carpeta **animals** y añadir ruta **animal/:id** con controlador **AnimalDetailsController**, vista **animal-details.html** y **controllerAs animal**
5. Crear fichero **animal-details.controller.js** dentro de la carpeta **animals** y definir controlador **AnimalDetailsController** y añadir propiedad **title** con cualquier valor
6. Crear fichero **animal-details.html** dentro de la carpeta **animals** y mostrar la propiedad **title** del controlador
7. Añadir dependencia al módulo **newHome.animals** al módulo principal **newHome**
8. Actualizar index.html con las referencias a los scripts de la carpeta **animals**

Ejercicio 10 - Añadiendo página de detalle

1. Abrir navegador y hacer clic sobre el nombre de un animal. Deberíamos ver la página de detalle de un animal mostrándonos el valor de la propiedad title que creamos en el controlador.

Ejercicio 11 - Obteniendo detalles de un animal

1. **git checkout 11get-animal-details --force**
2. El **endpoint** para leer los detalles de un animal es: **GET animals/1** donde 1 es el id del animal.
3. Añadir función **getById(id)** dentro del servicio **animalsService** que haga una llamada al endpoint para obtener un animal usando el servicio **$http** y la constante **API_ENDPOINT** y exponer dicha función
4. Inyectar servicios **$routeParams** y **animalsService** al controlador **AnimalDetailsController**
5. Crear propiedad vm.details en el controlador **AnimalDetailsController** que sea un objeto vacío
6. Crear función **activate()** en el controlador **AnimalDetailsController** que haga una llamada a la función **getById** del servicio **animalsService** pasando como parámetro el **id del animal** que se puede leer usando el serivio **$routeParams**

Ejercicio 11 - Obteniendo detalles de un animal

1. Cuando se resuelva la promise, guardar los datos del animal en la propiedad **vm.details**
2. Modificar **animal-details.html** para usar los datos que leemos en el controlador **AnimalDetailsController**
3. Abrir navegador y hacer clic sobre el nombre de un animal. Deberíamos ver todos los datos del animal.

Directivas

- Nos permiten extender HTML
- Aquí es donde se hace manipulación del DOM
- Angular trae varias como: ng-click, ng-model, ng-class, ng-show, ng-if, etc
- Tan simples como un contenedor o tan complejas como una mini aplicación
- Podemos (y debemos) crear nuestras propias directivas

Ejemplo 11

```
(function() {
    'use strict';

    angular
        .module('module')
        .directive('directive', directive);

    directive.$inject = ['dependencies'];

    function directive(dependencies) {
        var directive = {
            bindToController: true,
            controller: Controller,
            controllerAs: 'vm',
            templateUrl: 'template.html',
            link: link,
            restrict: 'A',
            scope: {
            }
        };
        return directive;

        function link(scope, element, attrs) {
        }
    }

    Controller.$inject = ['Controller'];

    function Controller() {

    }

})();
```

$scope de una directiva

- Puede usar el mismo $scope del controlador (scope: false)
- Puede tener su propio $scope que hereda del $scope del controlador (scope: true)
- Puede tener un $scope completamente aislado (scope: {})

Ejemplo 12

$scope aislado

- No hereda el $scope del controlador
- Útiles cuando creamos componentes reusables
- Pueden tener 3 tipos de propiedades
  - @: Permite que la directiva acceda al valor interpolado de un atributo del DOM
  - =: Enlaza una propiedad de una directiva a una propiedad del $scope padre
  - &: Permite ejecutar expresiones en el contexto del $scope padre

Ejemplo 13

Controlador de una directiva

- Exactamente igual que un controlador normal, pero en el contexto de una directiva
- Nos permiten controlar el estado de un componente

Ejemplo 14

Función link de una directiva

- Manipulación del DOM
- Registro de eventos
- Encapsulación de plugins de terceros
- Tiene 3 parámetros
  - scope: El scope de la directiva
  - element: Un objeto de jqLite o de jQuery que representa el elemento de la directiva
  - attributes: atributos del elemento donde definimos nuestra directiva

Ejemplo 15

Ejercicio 12 - Encapsulando información de un animal

1. **git checkout 12encapsulate-animal-card**
2. Crear carpeta **widgets** bajo **app**
3. Crear fichero **widgets.module.js** dentro de la carpeta **widgets** y definir módulo **newHome.widgets** sin dependencias.
4. Crear fichero **animal-card.directive.js** y definir directiva **nwhAnimalCard**, el template debe ser **animal-card.html**, el controlador **AnimalCardController,** controllerAs **card, bindToController** que contenga una propiedad **animal: '='** y por último el scope igualado a un objeto vacio
5. En el template **animal-card.html** de la directiva, poner el markup necesario para mostrar los datos del animal que que recibimos como parámetro.
6. Añadir dependencia al módulo **newHome.widgets** al módulo principal **newHome**

Ejercicio 12 - Encapsulando información de un animal

1. Modificar vista **home.html** y **animal-details.html** para que usen la directiva **nwh-animal-card** para mostrar los datos de un animal
2. Actualizar index.html con las referencias a los scripts de la carpeta **widgets**
3. Abrir navegador. Tanto en la homepage como en la página de detalle deberíamos seguir viendo la tarjeta con los datos de un animal.

Ejercicio 13 - Recuperando likes

1. **git checkout 13recovering-likes-feature --force**
2. Inyectar servicio **animalsService** al controlador **AnimalCardController** de la directiva **nwhAnimalCard**
3. **Mover** función **addLike** de **HomeController** al controlador **AnimalCardController** de la directiva **nwhAnimalCard**
4. Actualizar la plantilla **animal-card.html** de la directiva **nwhAnimalCard** y usar **ng-click** en el icono del corazón para llamar a la función **addLike** añadida anteriormente
5. Abrir el navegador. Cuando hagamos click en el icono del corazón, se debería incrementar el número de likes, tal como teníamos antes, pero ahora también funciona en la página de detalles de un animal.

Ejercicio 14 - Ocultando información de la tarjeta en la página de detalle

1. **git checkout 14hiding-card-info --force**
2. En la directiva **nwhAnimalCard** añadir la siguiente propiedad en el objeto ontroller: **showContent: '='bindToController: showContent: '='**
3. Esta propiedad representa si queremos mostrar el nombre y descripción de la tarjeta
4. Actualizar plantilla **animal-card.html** y usar **ng-show** para mostrar u ocultar el nombre y descripción del animal basándonos en el valor de la propiedad **showContent**
5. Ir a **HomeController** y agregar propiedad **showCardsContent** con valor **true**
6. Actualizar vista **home.html** para pasar el nuevo parámetro **show-content** a la directiva **nwh-animal-card**
7. Ir a **AnimalDetailsController** y agregar propiedad **showCardContent** con valor **false**

Ejercicio 14 - Ocultando información de la tarjeta en la página de detalle

1. Actualizar vista **home-details.html** para pasar el nuevo parámetro **show-content** a la directiva **nwh-animal-card**
2. Abrir el navegador. Al ir a la página de detalles de un animal, el nombre y la descripción en la tarjeta deberían estar ocultos.

Filtros

- Proveen formato a un valor que se muestra al usuario
- Se pueden usar en vistas, servicios, controladores y directivas
- Angular trae algunos como: date, currency, uppercase, limitTo, etc.
- Podemos crear nuestros propios filtros

Ejemplo 16

Ejercicio 15 - Formateando género de un animal

1. **git checkout 15formatting-animal-gender --force**
2. Crear carpeta **common** bajo **app**
3. Crear fichero **common.module.js** dentro de la carpeta **common** y definir módulo **newHome.common** sin dependencias.
4. Crear fichero **gender.filter.js** dentro de la carpeta **common** y definir un filtro llamado **gender**
5. El filtro debe devolver “Macho” o “Hembra”en función del género del animal, que llega en formato “M” para machos y “F” para hembras.
6. Añadir dependencia al módulo **newHome.common** al módulo principal **newHome**
7. Modificar vista **animal-details.html** para aplicar el filtro **gender**
8. Actualizar index.html con las referencias a los scripts de la carpeta **home**
9. Abrir el navegador. Cuando vayamos a la página de detalle de un animal deberíamos ver la información del género correctamente formateada.

Recursos (en inglés)

- Chuleta
- Guía de estilo
- Documentación oficial
- Wiki oficial
- Top 10 errores más comunes al empezar con Angular

¡Gracias!