Herramientas para el desarrollo.

TypeScript : Lenguaje de programación de Microsoft con el que se ha desarrollado angularjs 2.

Node.js: Entorno de ejecución javascript en servidor. Se instala node.js para usar la utilidad npm principalmente utilizado para realizar instalaciones.

Gulp:

Corre sobre Node.js, es un build system(sistema de construcción) que permite automatizar tareas comunes de desarrollo, tales como la minificación de código JavaScript, recarga del navegador, compresión de imágenes, validación de sintaxis de código y un sin fin de tareas más.

Para pruebas en local, puede ser interesante la funcionalidad localStorage de HTML5 usando formato JSON.

AngularJS

Conceptos

Módulo:

Se define como un contenedor de artefactos. Los artefactos definidos en Angular son:

* Controladores
* Directivas

Filtros

* Constantes
* Servicios
* Factorias
* Providers
* Config
* Run
* Bloques config y bloques run
* Etc

Lo normal no es crear un módulo y luego obtener una referencia a él sino hacerlo todo en la misma línea:

|  |  |
| --- | --- |
|  | var moduloAA=angular.module("AA",[]); |

Pero también podemos crear el módulo

angular.module("AA",[]);

y luego obtener una referencia a él:

var moduloAA=angular.module("AA");

bloque:

Un bloque config es una función que definimos en nuestro código pero que se ejecutará al iniciar un programa en AngularJS para configurar un [provider](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:09_provider" \o "unidades:03_servicios:09_provider). Se define llamando al método config de un módulo.

[?](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:04_tiposservicios)

Veamos un ejemplo:

|  |  |
| --- | --- |
| 1  2  3 | app.config(function() {    //Pon aquí el código que quieras  }); |

El bloque run es, al igual que el bloque config, una función que se ejecuta al iniciar un programa en AngularJS. Se define llamando el método run de un módulo.

App.run(function(){ …. Sentencias});

siendo app un módulo de Angular

La configuración de servicios requiere un provider y para configurar un provider necesitaremos un artefacto config. Los config se ejecutarán antes que los run.

Artefacto directiva:

Ng-model, para hacer referencia a los datos : ng-model=”seguro.id”

Ng-disable, para habilitar o deshabilitar campos html :

ng-disabled="seguro.enfermedades.alergia===false"

Artefacto servicio:

constant

* value
* service
* factory
* provider

Servicio

Definición

Un servicio es un objeto JavaScript que nos permite obtener información. Aparentemente nada nuevo que entender, sería por ejemplo un DAO en Java o un servicio de Java. Lo importante de ésto es que un servicio nunca [1)](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:01_servicios#fn__1) interacciona con la propia página, sólo con otros servicios o con un servidor de datos que pueda estar en otro Host.

Ejemplos de servicios serían:

* El servicio [$http](https://docs.angularjs.org/api/ng/service/$http) de AngularJS. Este servicio hace la típica llamada AJAX a un servidor para obtener información de él. Como vemos, cumple perfectamente la definición de obtener información.
* Un servicio que se conecta a un host que nos retorna el valor del Euribor.
* Un posible servicio de cálculo de Hipoteca que dados los datos de una hipoteca (Importe del préstamo, años, diferencial ,etc) nos calculara cuánto hay que pagar mensualmente.
* Un servicio que nos hiciera las operaciones de [CRUD](http://es.wikipedia.org/wiki/CRUD) sobre el servidor.
* Un servicio que transformara los String con una fecha en un objeto Date.
* El servicio [$log](https://docs.angularjs.org/api/ng/service/$log) de Angular que nos permite generar un log de nuestra aplicación.
* Etc.

Ventajas

**Las ventajas de los servicios son las siguientes:**

* Facilitar el TDD
* Permitir decorar los servicios
* Facilitar las relaciones entre servicios
* Modificar las implementaciones

Una característica importante de los servicios es que sólo hay una única instancia de ellos aunque los usemos más de una vez, es decir que son un [singleton](http://es.wikipedia.org/wiki/Singleton" \o "http://es.wikipedia.org/wiki/Singleton).

1. Facilitar el TDD

Aunque en este curso no se trata el tema del [Desarrollo guiado por pruebas](http://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas) o TDD, todo AngularJs está diseñado para facilitar el TDD. Un ejemplo de ello es que las instancias de los servicios no se obtienen directamente sino que se obtienen mediante [Inyección de Dependencias](http://cursoangularjs.es/doku.php?id=patrones:di). Es decir, en un controlador u otro servicio le diremos que deseamos una instancia de un servicio, pero no lo crearemos mediante la palabra reservada de JavaScript de new. Esto nos permitirá al hacer TDD modificar los servicios por objetos [Mock](http://es.wikipedia.org/wiki/Objeto_simulado" \o "http://es.wikipedia.org/wiki/Objeto_simulado) y poder hacer pruebas unitarias de nuestros controladores o servicios.

Es decir, al ser el servicio un artefacto de Angular , podremos cambiarlo por una implementación vacía que facilita el TDD.

1. Permitir decorar los servicios

Angular nos permite decorar los servicios. Ésto significa que permite que interceptemos la creación del servicio para poder añadir métodos os modificar su funcionamiento. Ésto es muy útil, por ejemplo, para poder hacer modificaciones en servicios de terceros que no podemos modificar.

1. Facilitar las relaciones entre servicios

Cuando la aplicación se vuelve compleja empiezan a depender unos servicios de otros. En los ejemplos de servicios que hemos visto en este tema, está el servicio de cálculo de hipoteca que dependería del servicio de obtención del euribor.

Cuando la cosa se complica podría pasar que se dependiera de un servicio que aún no ha sido inicializado. Angular tiene los 5 tipos de servicios que hemos enumerado formando una jerarquía que permite evitar esos problemas.

1. Modificar las implementaciones

Podríamos tener 2 módulos que implementen el mismo servicio aunque de 2 formas distintas. Cambiar de una implementación a otra sería tan sencillo como hacer que dependiera de un módulo u otro (Pero no de los 2 a la vez).

En nuestro ejemplo de servicio de obtención del Euribor se podría haber creado un módulo llamado “BBVA” que incluyera el servicio del euribor de forma que se conecte al Banco BBVA para obtener el valor del Euribor. En caso de que no nos interesara por cualquier motivo seguir usando el BBVA podríamos crear otro módulo llamado “Santander” en el que estuviera también el servicio de obtener el Euribor.

Cambiar de un servicio a otro sería tan sencillo como cambiar en el módulo de nuestra aplicación del módulo “BBVA” a “Santander”.

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

script.js (file)

app.controller("SeguroController",function($scope,$log) {

  $scope.seguro={

    nif:"",

    nombre:"",

    ape1:"",

    edad:undefined,

    sexo:"",

    casado:false,

    numHijos:undefined,

    embarazada:false,

    coberturas: {

      oftalmologia:false,

      dental:false,

      fecundacionInVitro:false

    },

    enfermedades:{

      corazon:false,

      estomacal:false,

      rinyones:false,

      alergia:false,

      nombreAlergia:""

    },

    fechaCreacion:new Date()

  }

  $log.debug("Acabamos de crear el $scope");

});

La primera característica de los servicios es que tienen un nombre (al igual que los controladores) y como ya dijimos en [Espacio de nombres](http://cursoangularjs.es/doku.php?id=unidades:02_angular:08_modulos#espacio_de_nombres) dicho nombre es independiente del módulo en el que fueron añadidos. Así que para usar un servicio sólo necesitamos saber su nombre.

También podemos declarar:

app.controller("SeguroController",['$scope','$log',function(a,b) {

.....

}]);

Los servicios al igual que las variables deben seguir nomenclatura camelCase, porque cuando inyectamos un servicio lo hacemos sobre una variable que se llama como el servicio, por lo tanto el servicio se debe llamar como las variables.

Servicio $timeout:

El servicio de [$timeout](https://docs.angularjs.org/api/ng/service/$timeout) es como el método [setTimeout()](http://www.w3schools.com/jsref/met_win_settimeout.asp" \o "http://www.w3schools.com/jsref/met_win_settimeout.asp) de JavaScript. La principal diferencia , a parte de que es un servicio, es que al igual que [3.2 $http](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:02_http)se actualiza la vista al actualizar el modelo desde $timeout.

En el módulo correspondiente tendremos que añadir:

$timeout(function() {

$scope.producidoEvento="SIIIIII";

$log.debug('finalizado');

},3000);

**¿Cuándo es posible inyectar los diferentes servicios?**

La diferencia principal entre ellos es dónde se pueden inyectar mientras se inicializa la aplicación, es decir que según el tipo de servicio los podremos inyectar en un bloque config o en un bloque run o en otro servicio, etc.

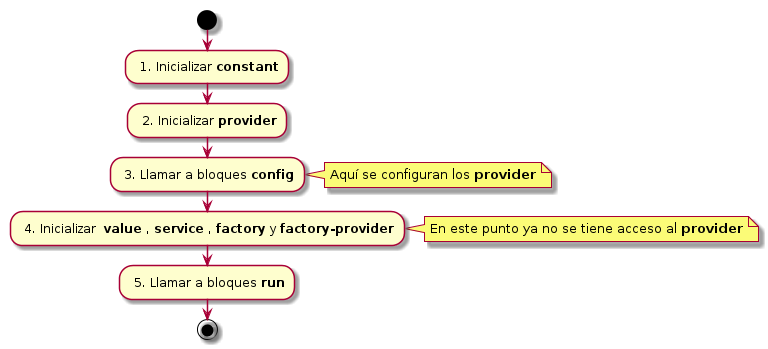
La siguiente tabla indica dónde se pueden inyectar o donde no inyectar cada uno de los servicios al **iniciar la aplicación**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Se puede inyectar en un** | | | |
| **Tipo** | **Provider** | **Bloque config** | **Servicio**[**1)**](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:04_tiposservicios#fn__1) | **Bloque run** |
| constant | Si | Si | Si | Si |
| provider | Si [2)](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:04_tiposservicios#fn__2) | Si | No | No |
| value | No | No | Si | Si |
| service | No | No | Si | Si |
| factory | No | No | Si | Si |
| factory-provider [3)](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:04_tiposservicios#fn__3) | No | No | Si | Si |

Una vez iniciada la aplicación, cualquiera de ellos, excepto el provider, ya se puede inyectar en cualquier otro artefacto como un controlador, directiva , etc.

Recordemos que un servicio provider es requerido para configurar otros servicios.

El siguiente diagrama de actividad explica los pasos que realiza AngularJS al iniciar una aplicación.

[](http://cursoangularjs.es/lib/plugins/plantuml/img.php?width=0&height=0&title=PlantUML%20Graph&align=&version=2011-07-16&md5=58be6a3decd72e1f33c6eb1a6c6b8396)

* Actividad 1: Se puede ver cómo lo primero que se hace es inicializar las constant, por lo tanto éstas se pueden inyectar en cualquier sitio.
* Actividad 2: Seguidamente se crean los provider
* Actividad 3: Se llaman a los bloques config. Por lo tanto en un bloque config se puede inyectar tanto unaconstant como un provider. En un bloque config lo que se hace es configurar los provider.
* Actividad 4: Ahora es cuando se llama a todos los servicios *normales* que son los value , service , factoryy factory-provider. Llegados a este punto los provider no están accesibles ya que al estar ya configurados solo se tiene acceso a los factory-provider correspondientes a sus provider.
* Actividad 5: Se llama al bloque run para inicializar la aplicación pudiendo inyectar cualquier tipo de servicio excepto los provider

Tipos:

Servicio constanst

A un servicio de tipo constanst, le podemos asignar valor y dicho valor podrá ser inyectado en cualquier parte de la aplicación.

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

app.constant("miServicioConstante","Hola mundo");

app.controller("PruebaController",["$scope","miServicioConstante",function($scope,miServicioConstante) {

  $scope.valor=miServicioConstante;

}]);

-.-.-.

  <body ng-controller="PruebaController">

    El valor de la constante "miServicioConstante" = {{valor}}

  </body>

Servicio value

Un value es un servicio al que le pasamos directamente el valor de dicho servicio. Se define llamando al método value de un módulo. A dicho método le pasaremos el nombre y su valor.

Servicio service

Hemos visto por ahora 2 tipos de servicio , las constant y los value. En ambos casos le pasábamos directamente el valor que debía tener el servicio. Con el tipo service [1)](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:07_service#fn__1) le debemos pasar una clase [2)](http://cursoangularjs.es/doku.php?id=unidades:03_servicios:07_service#fn__2) JavaScript y será AngularJS el que cree internamente una instancia de la clase.

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

app.value("tamanyoInicialRectangulo",{

  ancho:2,

  alto:3

});

function Rectangulo(tamanyoInicial) {

  this.ancho=tamanyoInicial.ancho;

  this.alto=tamanyoInicial.alto;

  this.setAncho=function(ancho) {

    this.ancho=ancho;

  }

  this.setAlto=function(alto) {

    this.alto=alto;

  }

  this.getArea=function() {

    return this.ancho \* this.alto;

  }

}

app.service("rectangulo",['tamanyoInicialRectangulo',Rectangulo]);

app.controller("PruebaController",["$scope","rectangulo",function($scope,rectangulo) {

  $scope.area=rectangulo.getArea();

}]);

Con app.service("rectangulo",['tamanyoInicialRectangulo',Rectangulo]);, será angularjs el que cree una instancia de la clase Rectangulo, y la que inyecte ‘tamanyoInicialRectangulo en el constructor.

Servicio factoria.

Cuando las usamos tienen la particularidad de devolvernos un dato, de cualquier tipo. Lo común es que nos devuelvan un objeto de Javascript donde podremos encontrar datos (propiedades) y operaciones (métodos). Con diferencia de los controladores, las factorías tienen la característica de ser instanciados una única vez dentro de las aplicaciones, por lo que no pierden su estado. Por tanto, son un buen candidato para almacenar datos en nuestra aplicación que queramos usar a lo largo de varios controladores, sin que se inicialicen de nuevo cuando se cambia de vista.

Necesidades que resuelven las factorías y no los controladores son:

1) Compartir datos entre varios controladores, lo que permite tener aplicaciones de verdad, capaces de memorizar estados entre varias de sus pantallas.   
  
2) Compartir datos entre varias vistas distintas. Por supuesto, sin usar las temidas variables globales. Eso es justamente lo que vimos en el artículo anterior, que ya tengas uno o varios controladores, no comparten ni se memorizan estados al pasar de una vista a otra.

Además, quizás el más representativo de los usos de las factorías es:

3) Empaquetar operaciones comunes a varios controladores (por ejemplo en una aplicación de facturación podríamos necesitar calcular el IVA o acceder a los distintos tipos de IVA en varios puntos del sistema). Por supuesto, no queremos colocar el código de esos cálculos u operaciones repetido en todos los controladores que deben utilizarlos y tampoco deseamos crear funciones con ámbito global.

Servicio provider

Test de aplicaciones angularjs:

Sobre una aplicación angularjs podemos hacer test unitarios y test E2E.

Herramientas para test unitarios Karma, JsTestDriver,

Herramientas para test E2E Protactor.

Karma y Protactor corren sobre Node.js

Los proyectos basados en Node.js, tiene un fichero de configuración llamado package.json.

El framework para desarrollar servicios REST sobre Node.js es ExpressJS, de manera análoga al framework Spring para java.

Comunicación entre controladores.

Existen tres tipos de comunicaciones.

**Mensaje para los controladores descendientes: $scope.$broadcast**

function padreCtrl($scope){

$scope.$broadcast('evento', [1,2,3]);

}

function hijoCtrl($scope){

$scope.$on('evento', function(event, data) { console.log(data)});

}

**Mensaje para el controlador ascendente: $scope.$emit**

function padreCtrl($scope){

$scope.$on('evento', function(event, data) { console.log(data)});

}

function hijoCtrl($scope){

$scope.$emit('evento', [1,2,3]);

**}**

**Para controladores que nos son ni padre ni hijo el uno del otro $rootScope.$broadcast**

function indexCtrl($rootScope){

$rootScope.$broadcast('evento', [1,2,3]);

}

Para gestionar los mensajes entre controladores se utiliza el servicio $on.

Dependencias entre módulos

|  |  |
| --- | --- |
|  | var moduloA=angular.module("A",[]);  var moduloB=angular.module("B",[]);  var app=angular.module("app",["A","B"]); |

De esta forma el modulo “app”, puede utilizar artefactos de los módulos “A” Y “B”

Para cargar los módulos dependientes, tendríamos que :

<script type='text/javascript' src='T.js'></script>

<script type='text/javascript' src='A.js'></script>

<script type='text/javascript' src='B.js'></script>

<script type='text/javascript' src='app.js'></script>

AngularJS no tiene referido a la carga de ficheros JavaScript sí hay otros proyectos que pueden ayudarnos como:

* [RequireJS](http://requirejs.org/)
* [angularjs-loader](https://github.com/hansl/angularjs-loader)

Un inconveniente es que los nombres de los artefactos no pueden coincidir dentro de diferentes módulos. Todo apunta a que finalmente se usará la nomenclatura FQCN de java para diferenciar los paquetes (com.antonio.controlador1).

Generadores de plantillas Angularjs.

Angular Seed

ngBoilerPlate

All in one Yeoman Angularjs Generator

Clever Stack

Mejora en las presentaciones de app angularJs.

**Lumix** (responsive). Librería que ofrece controles prefabricados como formulario, botones, iconos, …

**Angular-formly**. Librería que facilita la generación de formularios.

**Angular-ui**. Librería de componentes como grid , formularios,…

angular-grid-ui -> grid

ui-validate: validadores de formulario compatibles con angular-formly.

Angular-sublime-package: plugin para sublime text, que facilita el desarrollo ofreciendo alternativas para seleccionar métodos, funciones de las clases.

Angularjs-atom, plugin para Atom

Angularjs-brackets

**Angular-translate**

Seguridad.

JACSON WEB TOKE. JWT (JWT.IO), es un sistema que permite autenticación sin mantener una sesión. El token es generado y firmado por el servidor. Se entrega al cliente como respuesta al login y debe incluirse en cada petición del cliente. Esta codificado en base64.

Optimización de código

Grunt -> evita tareas repetitivas en javascript, como por ejemplo crear la carpeta de distribución el proyecto, minimizar/ofuscar el proyecto.

UglifyJS -> Minimizar el proyecto angularjs.

Batarang -> Batarang es una extensión de chrome para debug de aplicaciones angularjs.

Podemos modificar objectos del scope,

$scope.producto=”pan”

Para ver el cambio en los controles, $scope.$apply()