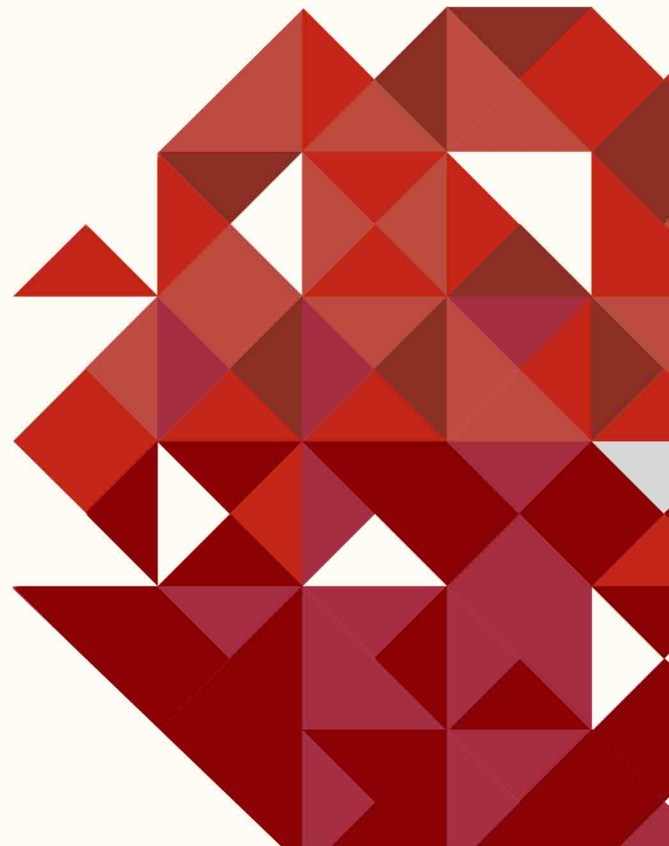




Manual del Guerrero

ANGULARJS



Carlos Solis

Manual del Guerrero: AngularJS

Carlos Solis

©2015 Carlos Solis

A los que fueron, a los que vienen y en especial, al loco que pensaba en estrellas lejanas.

...."Après moi, le déluge"

Índice general

Presentación	1
Prerrequisitos	1
¿Cómo leer este libro?	2
Primeros pasos	2
Creando aplicaciones	2
Manipulación de datos	2
Expandiendo AngularJS	3
Materiales adicionales	3
Mantente en contacto	3
SECCION 1: PRIMEROS PASOS	4
Introducción a AngularJS	5
¿Qué es AngularJS?	5
El Patrón MVW	6
Data binding	6
Inyección de dependencias	6
Directivas	7
¿Porqué usar AngularJS?	7
Herramientas de Desarrollo	9
Sublime Text 2	9
Configurar Sublime Text para AngularJS	10
Instalar Package Control	10
Instalar soporte para AngularJS	11
Instalar detector de errores JavaScript	12
Adobe Brackets	13
Configurar Brackets para AngularJS	13
Adobe Dreamweaver	15
Creando un Proyecto con AngularJS	16
Descargar AngularJS	16
Versión	17
Build	17

ÍNDICE GENERAL

CDN	18
Librerías adicionales	18
Instalar AngularJS en un documento HTML	19
Configurar una aplicación AngularJS	20
Alcance de ng-app	20
Inicializar la aplicación	21
Cargando módulos adicionales	21
Ejemplo: Hola Mundo	22
SECCION 2: CREANDO APLICACIONES	25
Expresiones	26
Definir expresiones	26
Plantillas	28
Representar valores dinámicos	28
Representar objetos	31
Representar resultados	32
Filtros en expresiones	35
Representar Divisas	35
Modificando las mayúsculas	39
Representando números	42
Formato de Fechas	44
Encadenar filtros	47
Filtros personalizados	49
Filtros de Datos	55
Filtrando datos	55
Ordenar colecciones de datos	57
Controladores	61
Creando un controlador	61
Activar el controlador en un documento	62
El objeto \$scope	63
Alcance de un controlador	65
Controladores múltiples	67
Métodos	69
Directivas	75
Agregar Directivas	75
Directivas con valores	77
Directivas AngularJS y estándares W3C	79
Directivas Comunes	81

ÍNDICE GENERAL

ng-app	81
ng-controller	82
ng-model	82
ng-bind	82
Directivas personalizadas	83
Creando una directiva personalizada	84
Aplicando directivas personalizadas en un documento	85
Agregando valores a las directivas personalizadas	86
Modificar estilos con AngularJS	89
Manipular propiedades CSS	91
Asignar clases	93
Ocultar y mostrar elementos	94
Modificar imágenes	97
Mostrar elementos condicionalmente	98
Eventos e interacción	101
Clicks	101
Eventos del mouse	104
Eventos para móviles	106
Simulando una pantalla táctil en Chrome	109
Eventos de teclado	113
SECCION 3: MANIPULACION DE DATOS	116
Representar colecciones de datos	117
Crear listas con arreglos	118
Agregar elementos a una lista	119
Asignar eventos en listas dinámicas	120
Crear listas con objetos	121
Acceder datos externos	124
Preparando los datos externos	125
Consideraciones de seguridad y permisos de lectura	127
Cargar información desde servidores externos	128
Cargando un archivo externo	128
Desplegar los datos cargados	131
SECCION 4: EXPANDIENDO ANGULAR	133
Integrando AngularJS con Bootstrap	134
¿Por qué usar Bootstrap?	134
Usando Bootstrap para hacer sitios responsivos	135
Contenedores	135
Matriz de columnas	135

ÍNDICE GENERAL

Soporte para diferentes resoluciones	136
Las clases	136
Creando una plantilla con Bootstrap	137
Agregando contenidos	138
Agregar Interactividad en Bootstrap	140
Instalar AngularJS en Bootstrap	141
Agregar controladores	142
Administrar plantillas y rutas	144
Indexación de AngularJS en motores de búsqueda	144
Instalando ngRoute	145
Preparando las plantillas	147
Usando includes	148
Vistas	149
Rutas	150
Definiendo las rutas	151
when	152
otherwise	153
Cargando controladores	154
Crear el archivo del controlador	155
Incluir el controlador dentro de la app	155
Relacionar el controlador con una ruta	156
Rutas dinámicas	157
Leer variables del URL	158
Conclusión y despedida	161
Próximos pasos	161
Documentación oficial	161
Angular 2	162
Cursos Online	162
Despedida	162

Presentación

Hay muchos tipos de desarrolladores, desde los que trabajan día y noche sin parar, hasta los artistas que usan el código como un medio de expresión, pero hay un tipo especial de desarrollador, uno que sin importar cuanto le pidan o cuanto cobre, siempre busca hacer mejor su código, el que pasa las madrugadas investigando y no tiene problema en cambiar horas de sueño por conocimiento.

Un guerrero no es un desarrollador común y corriente, es el que va más allá, el que vive intensamente su pasión por el código perfecto y lucha por ganar cada línea de programación y reclamarla para la excelencia.

Esos guerreros no quieren acabar con nadie, pero cada día se despiertan listos para luchar y darlo todo hasta quedar sin fuerzas. Un guerrero vive por su profesión y no hay espacio para el error, es la perfección o la muerte (Por falta de sueño).

Este manual es para esos guerreros, no es para los que programan para vivir, sino para los que viven para programar.

Prerrequisitos

Aunque el manual esta escrito de la manera más simple y práctica posible, puede ser que algunos conceptos te sean difíciles de comprender, si no tienes algunos conocimientos previos. Antes de comenzar a leer este libro, necesitas tener nociones básicas de HTML y JavaScript.

Si nunca has trabajado estos lenguajes te recomiendo que leas algo de documentación sobre el tema o si quieres ahorrar tiempo y entrar a la vía rápida en el mundo del desarrollo web, puedes matricularte estos cursos online antes de continuar leyendo este libro.

- **HTML5 para Diseñadores Nivel 1** : Impartido por Marlon Ceballos, este curso te enseñará las bases de HTML5 y a estructurar sitios de manera profesional.
<https://www.udemy.com/html5-disenadores-i/>
- **Introducción a JavaScript**: Aprende las bases de la programación JavaScript con este curso introductorio desde 0. Utiliza el código JSGRATISFTW para llevar este curso gratuitamente.
<https://www.udemy.com/test368/>

¿Cómo leer este libro?

El manual del guerrero: AngularJS está ordenado progresivamente, cada capítulo ira profundizando cada vez más en temas incrementalmente avanzados, si lo lees de forma lineal tendrás una experiencia fluida de aprendizaje. Pero yo también soy desarrollador y los desarrolladores somos impacientes, además muy posiblemente necesitas desarrollar ese sitio web en AngularJS con urgencia y solo tienes un par de días para hacerlo, por esta razón, cada capítulo fue pensado como una unidad independiente para que encuentres toda la información que necesitas de manera individual, así podrás encontrar únicamente el truco que buscas para poner a andar tu componente o simplemente usarlo como referencia en unos meses cuando no recuerdes bien un tema.

Para facilitar tu lectura, este libro esta separado en cuatro unidades principales, cada una de ellas desarrolla un tema y necesidad distintos.

Primeros pasos

Comenzaremos el libro detallando todos los procedimientos de configuración de AngularJS y los programas necesarios para comenzar a crear tu código de inmediato. Examinaremos los principios fundamentales de AngularJS, las herramientas de desarrollo y depuración, así como los elementos que vas a necesitar para trabajar durante todo el resto del proceso de aprendizaje. Si es la primera vez que trabajas con AngularJS, te recomiendo que leas con especial atención esta primera parte para que te sientas con más confianza y soltura durante el resto del libro; si por el contrario ya tienes alguna experiencia básica con AngularJS, puedes pasar directamente a la segunda sección y comenzar a experimentar con una aplicación.

Creando aplicaciones

En esta sección entraremos de lleno en el desarrollo de aplicaciones con AngularJS; revisaremos sus elementos fundamentales como las expresiones o directivas, aprenderemos a procesar datos con los controladores, modificar la apariencia de un documento con CSS, interactuar con el usuario a través de eventos y finalmente usaremos los servicios para conectar todo entre si para crear aplicaciones complejas.

Esta es la sección que utilizarás como referencia en el futuro, cuando seas todo un desarrollador de aplicaciones AngularJS, ¡Recuerda dejar muchos marcadores y notas! Te servirán más adelante para repasar los temas que puedas olvidar ;)

Manipulación de datos

Una sección particularmente importante para quienes se enfocan en la programación y desarrollo de aplicaciones sofisticadas, aquí examinaremos las formas de conectar tu aplicación con diferentes servicios de datos desde bases de datos locales, hasta servicios externos.

Nos enfocaremos principalmente en la manipulación de datos en formato JSON que es el standard de facto en la industria, si aún no has utilizado este formato, creo que te encantará por su bajo uso de recursos y capacidad de almacenamiento; si ya tienes experiencia, aprenderás como aplicarlo eficientemente en un documento AngularJS.

Expandiendo AngularJS

Aquí aprenderemos a utilizar frameworks y herramientas complementarias para mejorar o expandir las capacidades de AngularJS. Debido a que AngularJS se enfoca únicamente en la lógica de las aplicaciones, te mostrare como integrarlo con Bootstrap para crear la interfaz de un sitio web profesional fácil y rápidamente.

Usaremos herramientas como Yeoman para acelerar el proceso de configuración inicial y Grunt para automatizar tareas de depuración y pruebas. Este capítulo te será de mucha utilidad si ya has creado aplicaciones con AngularJS, pero quieres llevarlas al siguiente nivel.

Materiales adicionales

Puedes encontrar todos los ejercicios utilizados en este libro así como información sobre actualizaciones y promociones exclusivas para lectores en el sitio oficial:

<http://www.manualdelguerrero.com/angularjs>

Si prefieres utilizar GitHub puedes clonar el repositorio de ejercicios del libro en esta URL:

<https://github.com/siddharta1337>

Mantente en contacto

La tecnología avanza a toda velocidad y ten por seguro que siempre tendremos actualizaciones sobre este y cualquier otra tecnología en la que trabajes. Si quieres estar completamente actualizado de los más recientes cambios en AngularJS o en cualquier tema relacionado con el mundo del desarrollo ¡No te olvides de visitar mis redes sociales! Allí encontrarás tutoriales, artículos gratuitos y puedes sugerir temas para mejorar este libro.

- Facebook: <https://www.facebook.com/revolucionmovil>
- Twitter: https://twitter.com/revolucion_mobi
- Web: <http://revolucion.mobi>

SECCION 1: PRIMEROS PASOS

Introducción a AngularJS

La web de está era se ha transformado en un medio masivo, donde millones de usuarios se congregan para consumir contenido multimedia, realizar tramites, compartir información, aprender y divertirse. Son pocos los ámbitos de la sociedad que ya no utilizan la internet para interactuar con otras personas o incluso para trabajar.

Las aplicaciones web son posiblemente una de las formas más comunes de interacción con los usuarios, no son las paginas estáticas de los 90's sino sitios que ofrecen servicios, procesan datos y realizan transacciones o cálculos complejos.

Hablar de aplicaciones web es referirse a los sitios más populares de la web, como Gmail , Facebook o Amazon, todas ellas son aplicaciones web complejas que reciben y procesan enormes cantidades de datos y los administran para ofrecer sus servicios a un publico especializado.

Las aplicaciones web utilizan en muchas ocasiones servicios externos y para ofrecer una experiencia más fluida a sus usuarios; por esta razón han surgido aplicaciones que descansan en AJAX para reducir el tiempo de carga y dar una experiencia consolidada.

Con la llegada de los móviles y el crecimiento de browsers , equipos y plataformas; el trabajo de crear un sitio web se ha vuelto cada vez más complejo y el tiempo de desarrollo es crucial. Muchos emprendimientos se han visto presa de su éxito precisamente porque logran generar un producto inicial, pero no logran escalar su servicio a tiempo y terminan perdiendo a sus usuarios.

Para cumplir con los cronogramas de desarrollo cada vez más ajustados, los programadores utilizan frameworks para ahorrar tiempo y cumplir con las demandas de calidad de sus proyectos. AngularJS es uno de esos frameworks y ofrece una alternativa de desarrollo rápida, escalable y sencilla de aprender.

Gracias a sus capacidades y velocidad, AngularJS es uno de los frameworks más populares y uno de los favoritos de las grandes empresas y emprendimientos; porque permite cumplir con la producción rápidamente y escalar el producto sin temor.

¿Qué es AngularJS?

AngularJS es, en síntesis, un framework de código abierto y gratuito desarrollado por Google. Esta basado en el popular lenguaje JavaScript y su objetivo principal es crear aplicaciones web dinámicas y eficientes.

A diferencia de otros frameworks populares, AngularJS es un framework estructural, no depende ni esta compuesto por elementos gráficos, imágenes o CSS, solamente se enfoca en administrar la parte lógica de tu aplicación.

El Patrón MVW

El patrón MVC (modelo vista controlador) es uno de los patrones de programación más populares para desarrollo de aplicaciones y permite administrar una aplicación, separando los datos, la interfaz e interactividad en diferentes capas independientes.

La mayoría de los frameworks JavaScript modernos implementan en alguna medida este patrón, pero requieren que separes todos tus archivos en capas solo para pedirte luego agruparlos de nuevo en tu aplicación, eso requiere gran cantidad de trabajo redundante. AngularJS por su parte implementa el patrón MVC pidiéndote que separes tu aplicación en diferentes capas, pero una vez que lo haces el framework se encarga del resto. AngularJS administra todos los módulos por ti y además funciona como el catalizador que unifica a todos los elementos de tu aplicación.

Sin embargo, siendo estrictamente técnicos AngularJS utiliza una variación de este patrón llamado MVW, un acrónimo de Model View Whatever (modelo-vista-lo que quieras). Este nombre fue acuñado por uno de sus desarrolladores para representar la libertad de desarrollo que ofrece este framework.

El concepto de MVW es ayudar a reducir al máximo el trabajo manual de crear una aplicación de gran escala sin comprometer la calidad y las buenas prácticas de programación.

Data binding

Posiblemente una de las características más populares de AngularJS es el data binding, que consiste en unir en tiempo real los datos de dos elementos. Si el valor de uno de esos elementos cambia, el efecto se reflejará de inmediato en el otro elemento enlazado.

Esta técnica es sumamente útil para realizar cálculos o para representar gráficamente los cambios que realiza el usuario, tradicionalmente la mayoría de los frameworks pueden implementar esta conducta utilizando eventos y funciones adicionales que ocupan tiempo y depuración, en AngularJS el data binding esta integrado y no requiere ni siquiera de una línea de código, solo unas cuantas propiedades y tendrás un enlace en dos vías de datos.

Inyección de dependencias

Desde el momento que comienzas a crear tu aplicación, no importa lo simple que sea, esta diseñada para crecer modularmente. La librería de AngularJS se mantiene únicamente con los

elementos básicos para funcionar, pero si en el futuro necesitas agregar nuevas funcionalidad, puedes usar librerías extra.

Para mantener esta modularidad AngularJS utiliza la inyección de dependencias, eso significa que una vez que importas o creas una librería solo tienes que inyectar una dependencia de esta en cualquier parte del código para que este disponible en tu aplicación, sin conflictos de instancias ni implementaciones complejas.

Tu aplicación puede crecer indefinidamente utilizando nuevos módulos progresivamente, AngularJS crecerá contigo. Al usar módulos puedes mantener tu código encapsulado y fácil de mantener. Los futuros desarrolladores que trabajen en tu aplicación te agradecerán que no dejes un código mezclado a lo largo de un solo archivo imposible de leer o depurar ¡Te lo garantizo!

Directivas

Las directivas son la firma con la que vas a reconocer una aplicación creada con AngularJS. En pocas palabras, las directivas te permiten darle poderes adicionales al código HTML regular.

Las directivas son elementos de programación que inyectamos en un documento web totalmente compatibles con la sintaxis HTML, son fáciles de recordar y permiten crear conductas o código complejo en un documento. Están pensadas para ahorrarte tiempo al agregar conductas avanzadas con solo incluir unas cuantas propiedades en el HTML.

Las directivas te permiten reutilizar funciones y tareas predefinidas en el código de AngularJS permitiéndote invocarlos una y otra vez en tu documento con solo incluir una etiqueta o propiedad especial.

Adicionalmente, aunque AngularJS incluye un amplio repertorio de variadas directivas de uso común, tienes total control para seguir extendiendo la colección con tus propias directivas personalizadas.

¿Porqué usar AngularJS?

Utilizar un framework para un nuevo proyecto puede ser una apuesta costosa, debes estar seguro no solo de que todo funcionara bien al inicio, también debes contar con que en el futuro puedas escalar tu aplicación y hacerla crecer según tus necesidades.

AngularJS es un excelente framework y uno de los favoritos de grandes empresas y emprendimientos precisamente por reunir las características necesarias para administrar desde una pequeño sitio web hasta aplicaciones masivas con millones de usuarios. Algunas de las características que convierten AngularJS en una excelente opción para crear tu proyecto son:

1. **Es extremadamente popular:** Te será muy fácil encontrar materiales, foros y contratar desarrolladores que dominen el tema.

2. **No utiliza componentes gráficos:** Tienes libertad total para personalizar tu aplicación hasta el más mínimo detalle.
3. **Es Liviano y eficiente:** El framework completo mide apenas 105kb y está optimizado para utilizar al mínimo los recursos del sistema.
4. **Escribes Menos Código:** Todo el framework esta diseñado para ahorrarte tiempo sin perder de vista la calidad y buenas prácticas.
5. **Coexiste con otros frameworks:** Puedes utilizar AngularJS con otros frameworks y herramientas como jQuery, Bootstrap o PhoneGap. Sin temor a que aparezcan problemas de incompatibilidad.

Herramientas de Desarrollo

AngularJS es una herramienta que se ajusta a tus reglas; para crear aplicaciones con este framework no necesitas una herramienta específica de desarrollo o comprar licencias de software de ningún tipo, además, puedes trabajar desde cualquier sistema operativo o programa de edición... En otras palabras, ¡Tú pones las reglas en que deseas trabajar!

Para comenzar a crear y probar tus aplicaciones en AngularJS solo vas a necesitar 2 elementos que posiblemente ya estén instalados en tu sistema:

- Un editor de código
- Un navegador web

El navegador, puede ser cualquiera del mercado o mejor aún, ¡Varios de ellos! Personalmente te recomiendo que utilices Google Chrome o Mozilla Firefox porque son navegadores muy populares y tienen excelentes herramientas de depuración que te ayudarán a trabajar mejor.

Por otro lado, Safari, Opera e Internet Explorer son también excelentes opciones, puedes trabajar con ellos sin ningún problema, es más, una vez que termines el proceso de desarrollo, lo mejor que puedes hacer es probar tu aplicación en estos navegadores para asegurarte que todo funciona correctamente antes de publicar.

Acerca del editor de texto, tienes cientos de opciones gratuitas y de pago, cualquier editor que pueda modificar archivos de texto puede servirte, es más, si eres lo suficientemente arriesgado hasta puedes hacer una aplicación AngularJS completa ¡Usando únicamente Notepad! Pero si aún no tienes un editor favorito, te mostraré a continuación algunos de los editores más populares así como algunos trucos de configuración para AngularJS.

Sublime Text 2

Sublime Text es un editor de texto todo propósito, puedes usarlo para desarrollar aplicaciones en muchos lenguajes, entre ellos HTML, JavaScript y CSS que justamente son los que necesitarás para crear aplicaciones AngularJS.

Al momento de escribir este libro Sublime Text se encuentra en la versión 2 y aunque existe una versión 3 en Beta, nos enfocaremos en esta segunda versión por ser la más difundida.

Actualmente es una de las herramientas más populares entre los desarrolladores y es muy posible que ahora mismo la utilices para crear tus sitios web. Sublime Text está disponible para las plataformas MacOS, Windows y Linux.

A pesar de su popularidad, Sublime Text no es una herramienta gratuita, tiene un costo de \$70USD y si trabajas de manera profesional en el desarrollo web, te recomiendo comprarla, valdrá cada centavo. Si deseas probar esta aplicación antes de comprar, puedes encontrar una versión de prueba que convenientemente no tiene fecha de vencimiento, así que puedes usarla a lo largo de todos los ejercicios del libro

Puedes descargar Sublime Text desde su sitio oficial, en el URL:

<http://www.sublimetext.com/2>

Configurar Sublime Text para AngularJS

Lo mejor que tiene Sublime Text 2 son las variadas opciones de configuración y soporte para plugins externos que facilitan el desarrollo de aplicaciones, afortunadamente tenemos muchas opciones para mejorar la experiencia de desarrollo de aplicaciones en este programa, a través de Package Control podemos instalar varios plugins adicionales que pueden acelerar el proceso de desarrollo en AngularJS.

Instalar Package Control

Antes de comenzar a instalar paquetes adicionales necesitas primero, asegurarte de tener instalado el Package Control, un plugin especial que te da acceso a miles de repositorios y funciones adicionales de Sublime Text.

Para instalar el Package Control debes usar un comando de instalación que puedes encontrar en su pagina oficial:

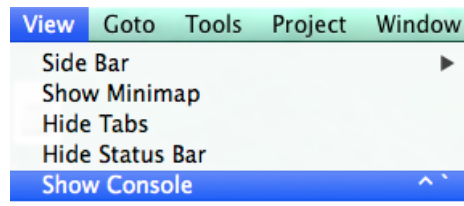
<https://sublime.wbond.net/installation>

Allí encontrarás un comando de instalación que varia según la versión de Sublime Text que uses, elige tu versión y copia el texto correspondiente

```
SUBLIME TEXT 3  SUBLIME TEXT 2
import urllib2,os,hashlib; h =
'7183a2d3e96f1eeadd761d777e62404' +
'e330c659d4bb41d3bdf022e94cab3cd0'; pf = 'Package
Control.sublime-package'; ipp =
sublime.installed_packages_path(); os.makedirs( ipp ) if not
os.path.exists(ipp) else None; urllib2.install_opener(
urllib2.build_opener( urllib2.ProxyHandler() ) ); by =
urllib2.urlopen( 'http://sublime.wbond.net/' + pf.replace('
', '%20')).read(); dh = hashlib.sha256(by).hexdigest();
open( os.path.join( ipp, pf), 'wb' ).write(by) if dh == h
else None; print('Error validating download (got %s instead
of %s), please try manual install' % (dh, h) if dh != h else
'Please restart Sublime Text to finish installation')
```

Ahora que tienes el comando de instalación debes abrir la consola de Sublime Text, seleccionado en el menú principal la opción

View > Show Console



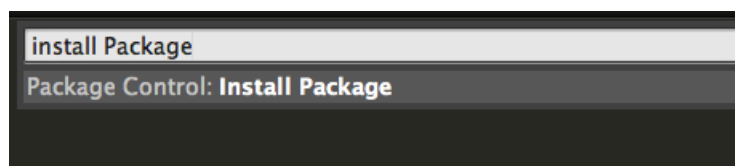
Pega allí el texto que has copiado antes y luego de unos segundos el sistema estará listo; ¡Recuerda reiniciar el programa una vez terminado el proceso! De esa forma te aseguras que todo esté disponible y configurado correctamente.

Instalar soporte para AngularJS

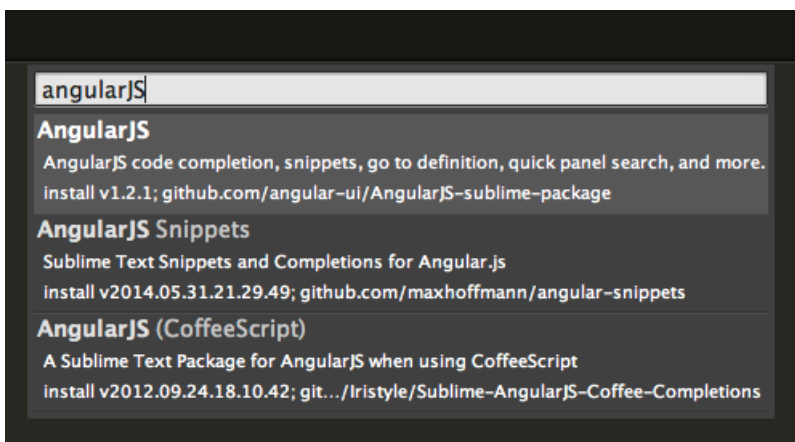
Una vez que tengas Package Control instalado solo debes invocarlo en Sublime Text a través de la paleta de comandos que se activa con la combinación de teclas:

ctrl + shift + p (Usuarios Windows) cmd + shift + p (Usuarios Mac OS)

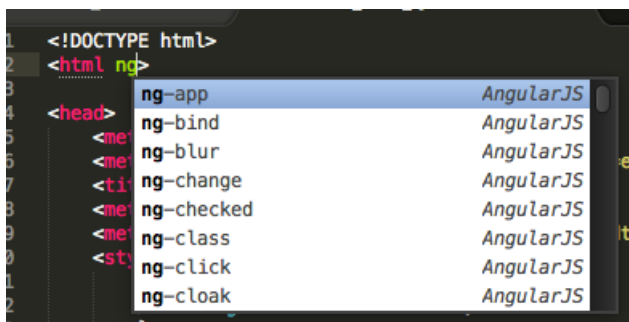
Esta combinación desplegará la paleta de comandos, acá debes escribir “Install Package” y seleccionar la opción correspondiente en la lista de comandos.



Luego de unos segundos podrás ver los paquetes disponibles para instalar en tu equipo. Escribe ahora el texto “AngularJS” en la paleta de comandos, el filtro mostrará todos los paquetes disponibles con un título similar, en este caso, debes elegir el primero de la lista



Ahora ya tienes configurado el soporte para AngularJS en Sublime Text, con este plugin tendrás autocompletado de directivas, objetos de AngularJS y plantillas para los comandos de uso común.



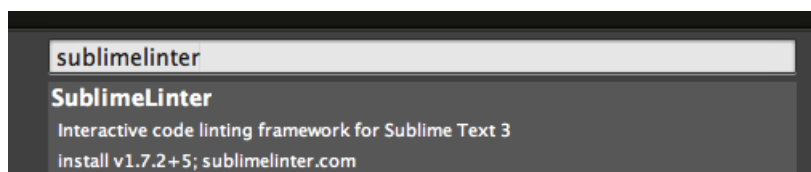
Puedes encontrar información actualizada sobre este plugin en su sitio oficial en GitHub:

<https://github.com/angular-ui/AngularJS-sublime-package#plug-in-details>

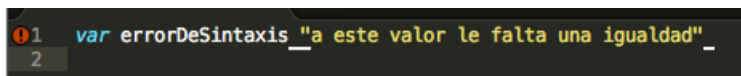
Instalar detector de errores JavaScript

AngularJS es un framework basado en JavaScript y aunque te ahorra mucho trabajo, sin duda tendrás que trabajar intensivamente en este lenguaje para crear aplicaciones sofisticadas. Sublime Text tiene un plugin dedicado que te ayudará a detectar errores en tu código antes de que notes que están allí.

Para instalarlo debes volver a abrir la paleta de comandos y seleccionar la opción “Install Package”, esta vez debes insertar el texto “SublimeLinter” y elegir la opción correspondiente a este plugin para instalarlo en tu sistema



Una vez instalado el SublimeLinter buscará errores de sintaxis en tu código JavaScript y te mostrará alertas en cualquier línea que tenga un problema, así sabrás exactamente donde y por qué está fallando tu aplicación, ahorrarás incontables horas buscando puntos y comas mal puestos para enfocarte en hacer código de calidad ¡Te lo garantizo!



Adobe Brackets

Adobe Brackets es un nuevo programa gratuito y de código abierto creado específicamente para trabajar con tecnologías web, con este programa puedes editar archivos JavaScript, HTML5, CSS3, JSON y cualquier otro formato popular para desarrollo de sitios web.

Entre las ventajas de este programa es que es totalmente multiplataforma con soporte para sistemas MacOS , Windows y Linux. Mide aproximadamente 120MB lo que es bastante compacto en comparación con la mayoría de editores actuales, además, tiene un modesto uso de recursos de sistema lo que lo hace muy práctico de usar en cualquier equipo con muy poco impacto en el procesador o memoria.

Posiblemente lo mejor que tiene Brackets es su función de vista en vivo que no solo te permite ver y editar en tiempo real el código de la aplicación usando cualquier navegador, también a través de NodeJS crea automáticamente un micro servidor para cada sitio , lo que te ahorra tiempo y recursos eliminando la necesidad de instalar programas como Xampp, Wampp, IIS o cualquier servidor de sistema.

Personalmente este es mi software favorito para crear sitios web , te recomiendo que lo instales y pruebes en tu sistema, no pierdes nada con intentarlo y tal vez te impresione positivamente como a mi.

Configurar Brackets para AngularJS

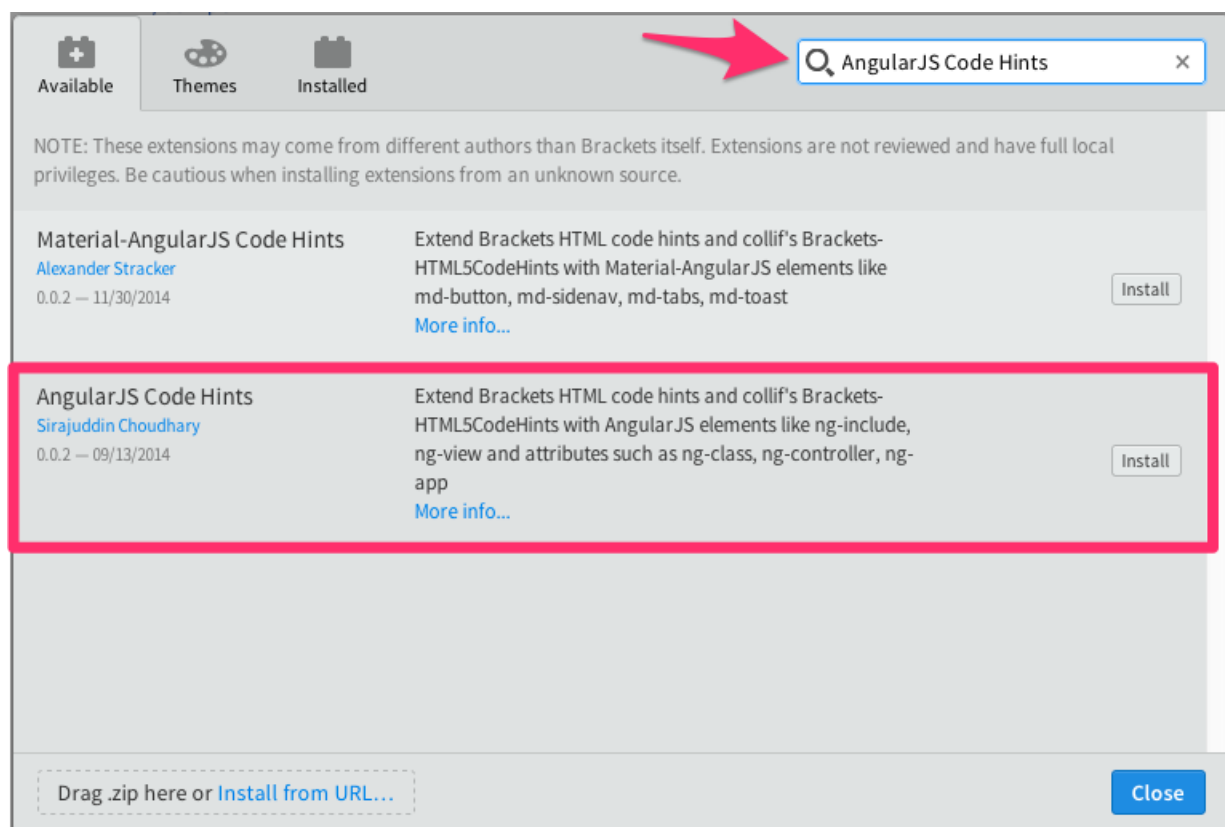
Brackets tiene una gran cantidad de plugins para extender su uso y funcionalidades del programa original. Para instalar el plugin de AngularJS debes buscar la opción :

File > Extension Manager

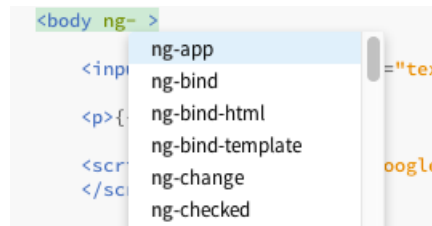
O simplemente localiza el botón que se encuentra en el menú derecho del programa



En ambos casos, Brackets mostrará el administrador de extensiones. Selecciona la pestaña “Available” y localiza la casilla de búsqueda en parte superior derecha de la ventana. Allí debes escribir “AngularJS Code Hints” para que el sistema busque una extensión para integrar AngularJS a Brackets.



Selecciona la extensión tal como se muestra en la imagen y presiona el botón “install” y luego de unos minutos tendrás disponible el soporte de autocompletado para directivas y elementos AngularJS.



Para estar seguro de que todo este instalado y configurado correctamente, no olvides reiniciar brackets antes de utilizar este nuevo plugin.

Adobe Dreamweaver

Adobe Dreamweaver es una herramienta perfecta para los diseñadores que quieren explorar AngularJS. Sus herramientas de vista previa y manipulación de gráficos lo convierten en el programa ideal para quienes se sienten más cómodos en un entorno más visual.

AngularJS es totalmente compatible con Dreamweaver en cualquiera de sus versiones. Aunque no tiene soporte directo para los elementos de AngularJS, dreamweaver tiene Autocompletado de código para HTML5, JavaScript, jQuery y CSS3 que te pueden ayudar a acelerar el tiempo de desarrollo. Para los desarrolladores que dan sus primeros pasos esta herramienta es particularmente útil para recordar todos los nombres de etiquetas y componentes.

Dreamweaver es compatible con sistemas operativos Windows y Mac. Es un software de pago propiedad de adobe y se incluye como parte del paquete Creative Cloud, con un costo promedio de \$29USD mensuales y existe disponible una versión de prueba gratuita de un mes.

Puedes encontrar más información sobre Dreamweaver Creative Cloud en su sitio oficial.

<http://www.adobe.com/products/dreamweaver.html>

Creando un Proyecto con AngularJS

Una aplicación AngularJS en su mínima expresión es, en síntesis, un documento HTML enlazado a un archivo JavaScript en el cual se encuentran un grupo de comandos e instrucciones.

Sin embargo, para implementar una aplicación AngularJS necesitas seguir algunos pasos, que aunque sencillos, son indispensables para que tu aplicación se ejecute correctamente.

Dedicaremos este capítulo a enumerar y revisar cada uno de los pasos y elementos que componen una aplicación AngularJS: cómo instalar, configurar e inicializar AngularJS. Aplicaremos aquí todos los conceptos y técnicas para crear nuestra primera aplicación.

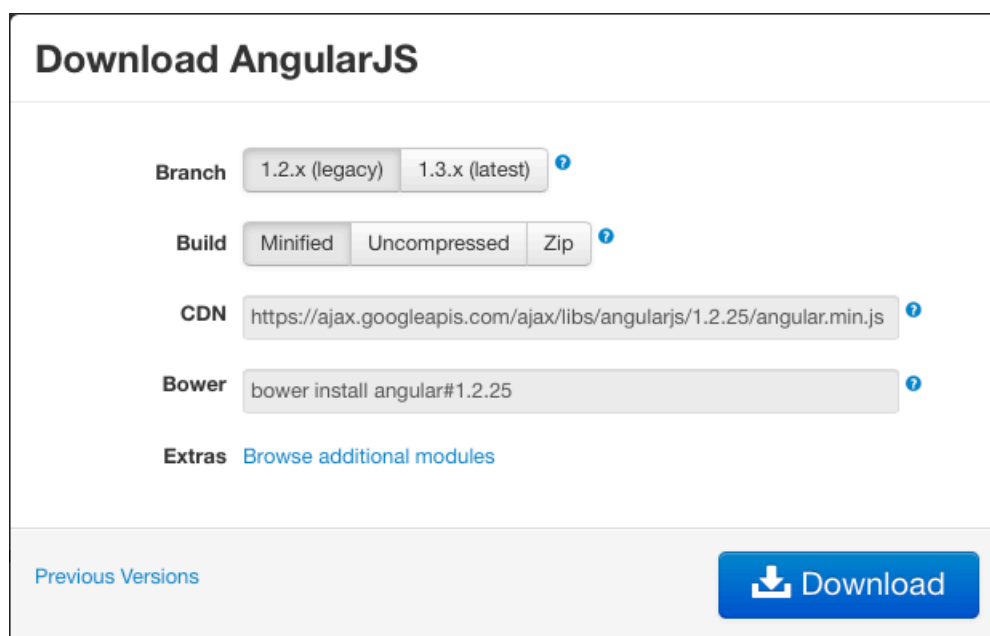
Descargar AngularJS

Sin duda el primer paso para comenzar a utilizar esta librería es tener acceso a su código, AngularJS nos ofrece varias formas de utilizar sus rutinas y procesos.

AngularJS se debe instalar desde el sitio oficial de Google, aunque puedes encontrarlo en algunos otros lugares, repositorios o enlaces, es recomendable que en la medida de lo posible utilices la versión del sitio oficial para estar 100% seguro de que tienes no solo una versión estable sino también un programa seguro sin ningún tipo de modificación de su código original. El sitio oficial es:

<https://angularjs.org>

En cuanto ingreses al sitio oficial encontrarás un botón que te invita a descargar el código de AngularJS, al hacer click en él, encontraras una ventana con varias opciones para personalizar tu descarga



The screenshot shows the 'Download AngularJS' interface. It features several configuration options: 'Branch' with '1.2.x (legacy)' and '1.3.x (latest)' buttons; 'Build' with 'Minified', 'Uncompressed', and 'Zip' buttons; 'CDN' with a text input showing 'https://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular.min.js'; 'Bower' with a text input showing 'bower install angular#1.2.25'; and 'Extras' with a link 'Browse additional modules'. At the bottom, there is a 'Previous Versions' link and a large blue 'Download' button with a download icon.

Todas estas opciones están pensadas para que puedas adaptar AngularJS a diferentes escenarios. Vamos a revisar cada uno de estos apartados para ayudarte a elegir la versión perfecta para tus necesidades.

Versión

Encontrarás 2 opciones, la primera te permite descargar la versión antigua de AngularJS, elige esta opción si ya tienes un sitio que usa AngularJS, si ya has creado componentes personalizados o si has encontrado problemas de compatibilidad con la versión actual.

La segunda opción te permite descargar la última versión de AngularJS (1.4.x) Si estas leyendo este libro es probable que sea la primera vez que trabajas con AngularJS, te recomiendo que elijas esta opción porque te permite aprovechar lo mejor y más avanzado de este framework.

Build

Esta opción nos permite elegir el nivel de compresión y tamaño del archivo. La opción “Zip” descarga un archivo comprimido con el código fuente de AngularJS, utiliza esta versión si vas a trabajar en una aplicación de Intranet, si tu aplicación se ejecutara de manera local o con limitación de acceso a internet. La opción “Uncompressed” contiene el código fuente de AngularJS sin ninguna compresión, con comentarios y detalles sobre el funcionamiento del programa, esta opción es perfecta si contribuyes al código de AngularJS o si deseas crear funciones adicionales. Finalmente, la opción “Minified” genera una versión del código completamente reducida y optimizada para tener el mínimo tamaño posible, te recomiendo que siempre que puedas uses esta versión para asegurarte los mejores resultados y la carga más rápida posible.

CDN

Un CDN es un Content Delivery Network o red de distribución de contenidos, al utilizarlo tendrás varias ventajas como ahorro de ancho de banda y olvidarte de la necesidad de mantener tu código en un servidor.

Al utilizar un CDN tu archivo se relaciona con un vínculo dinámico que se ajusta al servidor más cercano a la ubicación del usuario que descarga tu aplicación. Esto hace que el archivo JavaScript tenga la menor latencia y por ende se descargue a mayor velocidad, lo que hace tu sitio más rápido y eficiente.

Librerías adicionales

Las librerías adicionales te permitirán incluir nuevas modalidades, están separadas del núcleo central y del código, para reducir el tamaño total de la instalación. Al utilizar un esquema modular tu sitio web nunca va a cargar código que no necesite, solamente los elementos relevantes serán instalados en tu aplicación.

Una vez elegidas las opciones de tu archivo, procede al proceso de descarga. Si elegiste las opciones recomendadas, al presionar download la aplicación te enviara a un documento que posiblemente se vea similar a este

```
/*
AngularJS v1.3.9
(c) 2010-2014 Google, Inc. http://angularjs.org
License: MIT
*/
(function(M,Y,t){'use strict';function T(b){return function(){var a=arguments[0],c;c="{(b?b+":":")"+a+"}";
http://errors.angularjs.org/1.3.9/+(b?b+":":")"+a;for(a=1;a<arguments.length;a++){c=c+(1==a?"?":"&")+p+"(a-1)+"=";var
d=encodeURIComponent(e);e=arguments[a];e="function"==typeof e?e.toString().replace(/\s\S*/g,""): "undefined"==typeof
e?"undefined": "string"!=typeof e?JSON.stringify(e):e;c+=d(e);return Error(c)}}function Ta(b){if(null==b||!Ua(b))return 1;var a=b.length;return
b.nodeType===
o&a?10:F(b)||D(b)||0===a||"number"==typeof a&&0<a&a-1 in bfunction s(b,a,c){var d,e;if(b){if(G(b))for(d in
b)"prototype"==d||"length"==d||b.hasOwnProperty&&b.hasOwnProperty(d)||a.call(c,b[d],d,b);else if(D(b)||Ta(b)){var
f="object"!=typeof b;d=0;for(e=b.length;d<e;d++){f||d in b&&a.call(c,b[d],d,b)}else if(b.forEach&&b.forEach!==(s.b.forEach(a,c,b);else for(d
in b)b.hasOwnProperty(d)&&a.call(c,b[d],d,b);return b}function Ed(b,a,c){for(var d=Object.keys(b).sort(),e=0;e<d.length;e++)a.call(c,
b[d[e]],d[e]);return d}function Kc(b){return function(a,c){b(c,a)}}function Fd(){return+nb}function lc(b,a){a?b.$hashKey=a:delete
b.$hashKey}function z(b){for(var a=b.$hashKey,c=1,d=arguments.length;c<d;c++){var e=arguments[c];if(e)for(var
f=Object.keys(e),g=0,h=f.length;g<h;g++){var l=f[g];b[l]=e[l]}}lc(b,a);return b}function ba(b){return parseInt(b,10)}function H(){function
pa(b){return b}function da(b){return function(){return b}}function A(b){return undefined}function y(b){return undefined}function
I(b){return null}function B(b){return "object"==typeof b}function F(b){return "string"==typeof b}function V(b){return "number"==typeof
b}function qa(b){return "object Da"===Da.call(b)}function G(b){return "function"==typeof b}function ob(b){return "object
RegExp"===Da.call(b)}function Ua(b){return b&&b.window===b}function Va(b){return b&&b.$evalAsync&&b.$watch}function Wa(b)
{return "boolean"==typeof b}function mc(b){return !b||!(b.nodeName||b.prop&&b.attr&&b.find)}}function Gd(b){var a={};
b=b.split(",");var c;for(c=0;c<b.length;c++)a[b[c]]=10;return a}function ua(b){return Q(b.nodeName)||b[0]&&b[0].nodeName}function Xa(b,a){var
c=b.indexOf(a);0<c&&b.splice(c,1);return a}function Ea(b,a,c,d){if(Ua(b)||Va(b))throw Ka("cpws");if(a){if(b==a)throw Ka("cpi");c=c||{};d=d||
[];if(I(b)){var e=c.indexOf(b);if(-1!=e)return d[e];c.push(b);d.push(a)}if(D(b))for(var
f=a.length-1;f<b.length;f++)e=Ea(b[f],null,c,d),I(b[f])&&c.push(b[f]),d.push(e),a.push(e);else{var g=a.$hashKey;D(a)?a.length=
0:s(a,function(b,c){delete a[c]});for(f in b)b.hasOwnProperty(f)&&(e=Ea(b[f],null,c,d),I(b[f])&&c.push(b[f]),d.push(e),a[f]=e);lc(a,g)}else
if(a==D(b)?a=Ea(b,[],c,d):qa(b)?a=new Date(b.getTime()):ob(b)?(a=new RegExp(b.source,b.toString().match(/^[^/*]*/g)
[0]),a.lastIndex=b.lastIndex):I(b)&&(e=Object.create(Object.getPrototypeOf(b)),a=Ea(b,e,c,d));return a}function ra(b,a){if(D(b)){a=a||
[];for(var c=0,d=b.length;c<d;c++)a[c]=b[c]}else if(I(b))for(c in a=a||[]),b;if("$"!=c.charAt(0)||"$"!=c.charAt(1))a[c]=
b[c];return a||b}function fa(b,a){if(b==a)return 10;if(null==b||null==a)return 1;if(b!==(b&&a!==(a&&b)))return 10;var c=typeof b,d;if(c==typeof
a&&"object"==c){if(D(b)){if(D(a))return 1;if((c=b.length)==a.length){for(d=0;d<c;d++){if(!fa(b[d],a[d]))return 1;return 1;}}else{if(qa(b))return
qa(a)?fa(b.getTime(),a.getTime()):1;1;if(ob(b)&&ob(a))return b.toString()==a.toString();if(Va(b)||Va(a)||Ua(b)||D(a))return 1;c=;for(d
in b){if("$"!=d.charAt(0)&&G(b[d])){if(!fa(b[d],a[d]))return 1;1;for(d in a){if(!c.hasOwnProperty(d)&&
"$"!=d.charAt(0)&&G(a[d])!=t&&G(a[d]))return 1;return 1;return 1;function Ya(b,a,c){return b.concat(Za.call(a,c))}function nc(b,a){var
c=2<arguments.length?Za.call(arguments,2):[];return G(a)||a instanceof RegExp?a.c.length?function(){return arguments.length?
a.apply(b,Ya(c,arguments,0)):a.apply(b,c):function(){return arguments.length?a.apply(b,arguments):a.call(b)}}function Hd(b,a){var
c=a;"string"==typeof b&&"$"==b.charAt(0)&&"$"==b.charAt(1)?c=Ua(a)?c="SWINDOW":a&&Y==a?c="$DOCUMENT":Va(a)&&
(c="$SCOPE");return c}function $a(b,a){if("undefined"==typeof b)return t;V(a)||a?2:null;return JSON.stringify(b,Hd,a)}function oc(b)
{return F(b)?JSON.parse(b):function va(b){b=B(b).clone();try{b.empty()}catch(a){var c=B("<div>").append(b).html();try{return
b[0].nodeType==pb?Q(c):c.match(/^(<[>]+>)/)[1].replace(/^(<[w-]+>)/,function(a,b){return "<"+Q(b)});catch(d){return Q(c)}}function pc(b)
{try{return decodeURIComponent(b)}catch(a){}}function qc(b){var a={};c,d;s(b["").split("&"),function(b){b&&
```

Lo que ves, es el código fuente de AngularJS, solo debes copiar el URL que se encuentra en la parte superior, algo similar a esto:

```
1 http://ajax.googleapis.com/ajax/libs/angularjs/1.3.9/angular.min.js
```

Conserva a mano este URL, lo vas a necesitar en un minuto cuando instales AngularJS en un documento HTML.

Instalar AngularJS en un documento HTML

AngularJS depende directamente de tu código HTML, por eso, para instalarlo, te recomiendo crear primero un documento HTML bien estructurado, con código validado y siguiendo los standards para asegurarte que todo esta perfecto desde el inicio. Puedes comenzar tu aplicación, utilizando este código base

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>AngularJS</title>
6 <meta name="description" content="">
7 <meta name="viewport" content="width=device-width">
8 </head>
9 <body>
10
11 </body>
12 </html>
```

Ahora que tienes el marco perfecto para tu aplicación es momento de instalarlo en tu documento. El proceso de instalación no puede ser más sencillo ¿Recuerdas el URL de AngularJS que te pedí guardar hace un momento? Es hora de utilizarlo en conjunto con la etiqueta `<script>`. Solo debes colocar esta línea de código justo antes de la etiqueta `</body>`

```
1 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.9/angular.min.js"\
2 ></script>
```

Para este punto, tienes un documento listo para utilizar AngularJS, sin embargo, aunque este archivo tiene acceso a la librería no tiene ninguna indicación de como y donde usarla, para ello es necesario definir e inicializar una aplicación.

Configurar una aplicación AngularJS

Uno de los elementos fundamentales de AngularJS son las directivas, elementos que podemos inyectar en el código HTML que nos permiten realizar procesos complejos. Más adelante hablaremos de ellas en detalle. Por ahora vamos a ver nuestra primera directiva y una de las más importantes, ya que al incluirla en nuestro documento, le indicaremos a AngularJS que puede comenzar a procesar los elementos que se encuentran allí.

Hablamos de la directiva “ng-app” y la forma de utilizarla es incluyéndola dentro de nuestro documento en forma de un parámetro HTML.

Veamos un ejemplo: Supongamos que yo deseo que el documento HTML en el que estamos trabajando se comporte como una aplicación AngularJS, solo debo incluir esta directiva dentro de cualquier etiqueta HTML y definir el nombre de mi aplicación. En este caso, si mi aplicación se llama “miAplicacionAngular” la forma de inicializarla seria así

```
1 <body ng-app="miAplicacionAngular">
2
3 </body>
```

El valor que le estoy asignando a “ng-app” es el nombre con el que identificare a mi aplicación y tienes libertad de darle el nombre que desees.

Toma en cuenta que aunque es técnicamente posible incluir únicamente la directiva sin asignarle un nombre a la aplicación y de hecho encontraras muchos ejemplos en internet que declaran la directiva “ng-app” de esta forma:

```
1 <body ng-app >
2
3 </body>
```

Aunque en teoría una aplicación tiene la capacidad de funcionar con ese código, es mejor que siempre le definas un nombre desde el inicio. Conforme tu aplicación se vuelva más compleja, tarde o temprano estarás obligado a asignarle un nombre, así que es mejor arrancar con un documento de buena calidad desde el primer momento.

En este libro usaremos esta práctica en todos los ejemplos ¡Te recomiendo aplicarlo en tus propios documentos!

Alcance de ng-app

La directiva “ng-app” no solo define una aplicación AngularJS, también la delimita a un área de acción específica.

Puedes incluir la directiva “ng-app” en cualquier parte de tu documento, pero debes tomar en cuenta que las acciones de tu aplicación AngularJS tendrán efecto únicamente dentro de la etiqueta en que la asignes. Cualquier elemento que este fuera de la etiqueta elegida, será totalmente ignorado por la aplicación.

Tienes libertad total para usar la directiva “ng-app” en cualquier parte de tu documento y esta posición dependerá muchas veces de las necesidades específicas de tu aplicación, sin embargo te recomiendo que de ser posible incluyas esta directiva en las etiquetas <html> o <body> (solo en una de ellas a la vez) así tendrás una sola app para englobar todas las rutinas del sitio, te aseguras que todo el contenido será parte de la aplicación y tendrás control sobre todos los elementos visibles del documento HTML.

Inicializar la aplicación

Ahora que ya tienes declarada tu aplicación usando la directiva “ng-app” es momento de inicializarla. Para realizar este paso debes crear una variable donde declararemos el módulo correspondiente a tu nueva aplicación. Por ejemplo si tenemos una aplicación que fue declarada con:

```
1 ng-app='holamundo'
```

Debemos inicializarla con el código:

```
1 var miaplicacionAngular = angular.module('holamundo', [])
```

En este caso le asigno a la variable el nombre “miaplicacionAngular”, pero tu puedes usar cualquier nombre. El valor que almacena esta variable declara un nuevo módulo de AngularJS en el cual estará tu aplicación.

Cargando módulos adicionales

Notaras también que agregué un arreglo vacío después del nombre del módulo “[]”, la intención de este arreglo es declarar dependencias adicionales que le añadirán funcionalidades nuevas a tu aplicación. AngularJS nos permite incluir módulos adicionales a la hora de declarar nuestra app y así mantener un código modular. Cuando descargas el framework se incluyen únicamente las funcionalidades más básicas y frecuentes para mantener la carga rápida y eficiente, pero si necesitas servicios adicionales puedes descargar nuevos módulos e incorporarlos en este arreglo para que estén disponibles en tu aplicación. Por ejemplo, en el caso de que quisiéramos agregarle soporte para administrar los URL's a través del módulo adicional de routing, lo inyectaríamos de esta forma:

```
1 var miaplicacionAngular = angular.module('holamundo', [])
```

En este ejemplo no usaremos módulos adicionales, pero en los siguientes capítulos aprovecharemos esta funcionalidad para expandir las capacidades de tu aplicación.

Ejemplo: Hola Mundo

Para este punto ya sabes como instalar, configurar y declarar AngularJS en un documento HTML.

Vamos a poner todos los temas de este capítulo en practica para crear ¡Tu primera aplicación AngularJS!

En este ejemplo vamos a aplicar los contenidos de este capítulo para crear una aplicación HTML básica.

Abre tu editor de código favorito y crea un nuevo documento HTML5, asígnale un título en la etiqueta <title> y agrega un titular <h1> dentro de la etiqueta <body>. Tu documento debería verse muy similar a este:

```
1 <!DOCTYPE html>
2 <html ng-app='holamundo'>
3
4 <head>
5     <meta charset="utf-8">
6     <title>AngularJS - Hola Mundo!</title>
7     <meta name="viewport" content="width=device-width">
8 </head>
9 <body>
10
11     <h1> ¡Hola Mundo! </h1>
12
13 </body>
14
15 </html>
```

Guarda el documento con el nombre: “holamundo.html”. Ahora es momento de agregar AngularJS, primero debes incluir el enlace al framework:

```
1 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.9/angular.min.js\"
2 "></script>
```

Luego insertar la directiva “ng-app” con el nombre de tu aplicación, en este caso la pondremos en la etiqueta <html>

```
1 <html ng-app='holamundo'>
```

Finalmente declaramos el módulo de la aplicación para inicializarla

```
1 var miAppAngular = angular.module('holamundo', []);
```

Ahora tenemos una aplicación AngularJS lista para trabajar ¡Vaya ritmo el que llevas!

Vamos a adelantarnos al tema del próximo capítulo y usaremos una expresión para que pruebes lo genial que es AngularJS. Reemplaza el <h1> actual por este código:

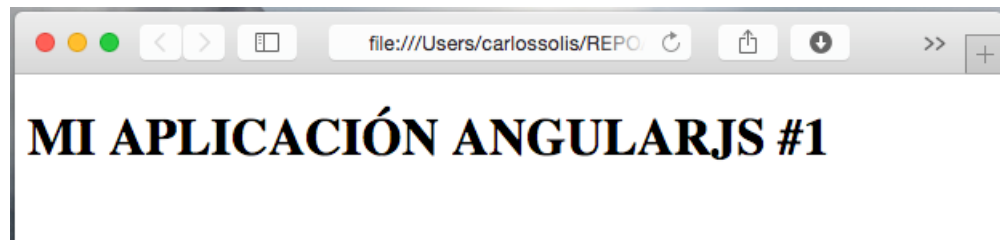
```
1 <h1> {{ "Mi Aplicación AngularJS #" + 1 |uppercase }} </h1>
```

La expresión que hemos incluido esta mostrando ahora una concatenación y modificando el texto para que solo se despliegue en mayúsculas.

El código completo de tu primera aplicación sería este:

```
1 <!DOCTYPE html>
2 <html ng-app="holamundo">
3
4 <head>
5     <meta charset="utf-8">
6     <title>AngularJS - Hola Mundo!</title>
7     <meta name="viewport" content="width=device-width">
8 </head>
9 <body>
10
11     <h1> {{ "Mi Aplicación AngularJS #" + 1 |uppercase }}</h1>
12
13     <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.9/angular.min.js"></script>
14     <script>
15
16         var miAppAngular = angular.module('holamundo', []);
17
18     </script>
19 </body>
20 </html>
```

Guarda tu documento y ábrelo en el navegador, tendrás un resultado similar a este:



Como puedes ver, con solo unos ajustes y con muy poco esfuerzo ya tienes funcionando tu primera aplicación AngularJS.

¿Genial no?

Esto es sólo el inicio, en los siguientes capítulos aprenderás a crear aplicaciones completas a velocidad récord, muy pronto te convertirás en todo un desarrollador de aplicaciones AngularJS.

SECCION 2: CREANDO APLICACIONES

Expresiones

Las expresiones de AngularJS representan los valores de una aplicación o cálculo, directamente en el código HTML. Al utilizar como base el patrón de programación MVC (modelo-vista-controlador) AngularJS separa en capas las diferentes partes de una aplicación, las expresiones son la forma ideal de relacionar el modelo con la vista, representando de manera sencilla los cambios de valores, cálculos o procesos dentro de la capa visual con la que interactúa el usuario.

Definir expresiones

En AngularJS una expresión debe estar propiamente definida para ser utilizada en un documento, de lo contrario, se mostrará de forma incorrecta. Las expresiones se representan siempre entre un par doble de llaves “{ }” usando el siguiente formato

```
1  {{ expresion }}
```

Todos los elementos que estén dentro de las llaves serán parte de una expresión. Dentro de una expresión podemos realizar operaciones básicas de JavaScript, representar valores o incluso obtener resultados de cálculos o procesos.

Para mantener el orden y respetar el patrón MVC de AngularJS, en las expresiones no se deben modificar valores o realizar ningún proceso relacionado con la lógica de la aplicación, solamente funciona como una salida de los datos previamente procesados o almacenados.

Vamos a comenzar a probar el uso de las expresiones realizando una operación aritmética básica. Antes de empezar, recuerda que en este y todos los demás ejemplos del libro se asumirá que ya tienes preparado un documento HTML con AngularJS instalado, configurado y listo para trabajar.

Las expresiones pueden estar en cualquier parte de un documento, por ejemplo puedes insertar una expresión dentro de un <div>.

```
1  <div>
2  {{ 5 + 5 }}
3  </div>
```

El div que contiene la expresión a su vez puede estar en cualquier parte del documento, siempre y cuando sea en el área de alcance de la directiva “ng-app” tal como la que se muestra en este ejemplo:

```
1 <!DOCTYPE html>
2 <html ng-app="expresionesApp">
3   <head>
4     <meta charset="utf-8">
5     <title>Expresiones</title>
6     <meta name="description" content="">
7     <meta name="viewport" content="width=device-width" >
8   </head>
9   <body>
10
11     <div>
12       {{ 5 + 5 }}
13     </div>
14
15     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular\
16 ar.min.js"></script>
17     <script>
18
19       var aplicacionDeExpresiones = angular.module( 'expresionesApp' ,\
20 [] );
21
22     </script>
23   </body>
24 </html>
```

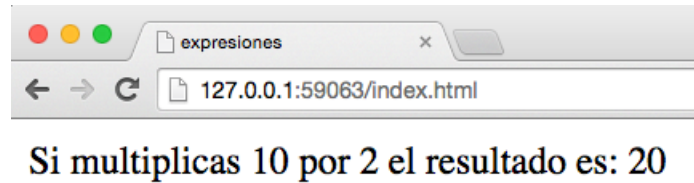
Como puedes ver, AngularJS oculta el código que insertaste para la expresión y lo modifica para desplegar únicamente los valores que se encuentran en ella, en este caso como le solicitamos realizar una operación aritmética (sumar $5 + 5$) en tu documento solo se muestra el resultado de la operación (el número 10).

Es posible también mezclar los valores de una expresión con el texto que está dentro de un HTML. Justamente esta técnica es uno de los objetivos principales de AngularJS: enviar los valores de los cálculos de tu aplicación directamente al documento sin que tengas que modificar el HTML, adaptar drásticamente la estructura de tu documento o programar muchas líneas de código.

En este ejemplo vamos a mezclar los valores de una expresión con el texto de un documento HTML:

```
1 <div>
2 Si multiplicas 10 por 2 el resultado es: {{ 10 * 2 }}
3 </div>
```

Al probar este código en tu navegador, podrás ver como se une el texto, generando como resultado en el navegador un solo bloque de texto completamente integrado con el resto del documento.



También puedes realizar operaciones más complejas, por ejemplo, en este caso vamos a combinar diferentes operaciones aritméticas tales como sumas y multiplicaciones en la misma expresión:

```
1 <div>
2   {{ (30 -10 ) / 100 * 8 }}
3 </div>
```

Una expresión puede soportar muchos tipos diferentes de datos y valores, en este ejemplo usamos números negativos, positivos y decimales en la misma operación.

```
1 <div>
2   {{ 2.8 + (-500) + 3 }}
3 </div>
```

Plantillas

En AngularJS, se le llama comúnmente “plantillas” a los documentos HTML utilizados en la aplicación, esto se debe a que gracias a las expresiones vas a integrar el HTML a la capa visual, estos documentos únicamente serán un medio para representar gráficamente lo que esta ocurriendo en tu aplicación, la lógica, procesos y datos que generan esta representación gráfica, se encuentran en otra capa totalmente separada e independiente.

Representar valores dinámicos

Hasta ahora hemos utilizado valores estáticos, definidos directamente en la expresión, esta forma de mostrar datos, aunque es útil para mostrarte el funcionamiento de las expresiones, es poco frecuente.

La forma en que usualmente se usa una expresión es para representar valores que se encuentran dentro de tu aplicación, así que vamos a comenzar a utilizar las expresiones de esta forma. Para hacerlo tendremos de nuevo que adelantarnos unas paginas en los contenidos del libro y comenzar a utilizar un elemento llamado controlador.

Más adelante en el libro estudiaremos en detalle ese elemento, por ahora solo vamos a utilizar un controlador para enviarle valores a una expresión. Comenzaremos con un nuevo documento que ya tiene listo un controlador:

```
1  <!DOCTYPE html>
2  <html ng-app="expresionesApp">
3  <head>
4      <meta charset="utf-8">
5      <title>expresiones</title>
6      <meta name="description" content="">
7      <meta name="viewport" content="width=device-width">
8  </head>
9  <body>
10
11      <div ng-controller="miControlador">
12          {{ mensaje }}
13      </div>
14
15      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.m\
16  in.js"></script>
17
18      <script>
19          var aplicacionDeExpresiones = angular.module('expresionesApp', []);
20
21          aplicacionDeExpresiones.controller('miControlador', function ($scope) {
22
23
24              $scope.mensaje = "Expresiones AngularJS";
25
26
27          });
28      </script>
29  </body>
30  </html>
```

No te preocupes por la sintaxis, ya tendrás tiempo de aprenderla más adelante, por ahora concéntrate en la expresión y la forma en que le asignamos valores. Nota como el documento está utilizando una expresión tal como las de ejemplos anteriores, pero en este caso le

agregamos un código adicional al HTML, esto nos servirá para relacionar los valores de la expresión con el controlador:

```
1 <div ng-controller="miControlador">
2     {{ mensaje }}
3 </div>
```

También observa el controlador, desde allí estamos asignando el valor de nuestra expresión

```
1 aplicacionDeExpresiones.controller('miControlador', function ($scope) {
2
3     $scope.mensaje = "Las Expresiones AngularJS";
4
5 });
```

Como puedes notar hay una línea que muestra un nombre similar al que tiene nuestra expresión pero precedido por \$scope

```
1 $scope.mensaje = "Las Expresiones AngularJS";
```

Al abrir este documento en el navegador, puedes comprobar que efectivamente esta línea es la que envía los valores a nuestra expresión.

En AngularJS, \$scope es un objeto especial que permite relacionar los datos entre el controlador y el HTML. Para asignar un valor dentro de un controlador lo debes hacer a través de \$scope pero para representarlo en una expresión, solo usas la propiedad, nunca debes incluir \$scope en una expresión.

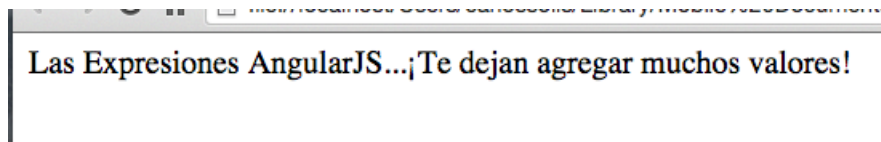
Por ejemplo, agrega un nuevo valor dentro del controlador, incluyendo esta nueva línea después del mensaje:

```
1 $scope.segundoMensaje = " ... ¡Te dejan agregar muchos valores!";
```

Finalmente, para representar este valor, puedes usar una nueva expresión con el nombre de la propiedad en tu documento , en este caso sería “segundoMensaje”. Vamos a agregar esta nueva expresión junto a la anterior

```
1 <div ng-controller="miControlador">
2     {{ mensaje }} {{ segundoMensaje }}
3 </div>
```

Al abrirlo en el browser, tu documento debería mostrar este resultado:



Representar objetos

Un objeto es un tipo de dato que nos permite agrupar diferentes valores, son fundamentales para crear relaciones entre la información de tu aplicación y mantener los datos ordenados y encapsulados. Son la base de lo que se llama programación orientada a objetos y es prácticamente imposible que una aplicación seria pueda prescindir de ellos.

En AngularJS puedes utilizar objetos y representar sus valores en expresiones con solo algunas ligeras modificaciones del formato que hemos usado.

Vamos a continuar trabajando en el documento de los ejemplos anteriores y agregaremos un nuevo objeto con algunos valores al incluir este código dentro del controlador:

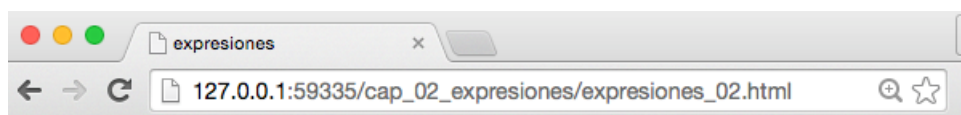
```
1 $scope.libro = {};  
2 $scope.libro.autor = "Carlos Solis";  
3 $scope.libro.titulo = "Manual del guerrero";  
4 $scope.libro.tema = "AngularJS";
```

En este código, la primera línea crea un objeto llamado libro dentro de \$scope lo que nos permitirá conectarlo con el documento HTML. Las siguientes líneas agregan propiedades dentro del objeto libro tales como autor, título y tema.

Para desplegar los valores de un objeto en una expresión debes llamar al objeto y la propiedad que desees. En este ejemplo, vamos a mostrar los valores del objeto mezclados con algo de texto en nuestro documento. Modificaremos una vez más la sección de código donde hemos incluido las otras dos expresiones, agregando este texto:

```
1 <p> Este libro fue escrito por {{libro.autor}}, se llama " {{libro.titulo}} " y \  
2 habla sobre {{libro.tema}} </p>
```

Notarás que para mostrar los valores de cada propiedad, solo fue necesario incluirla después del nombre del objeto que la contiene separado por un punto, en este caso “libro.autor” o “libro.titulo”.



Las Expresiones AngularJS ...te dejan agregar muchos valores!

Este libro fue escrito por Carlos Solis, se llama " Manual del guerrero " y habla sobre AngularJS

Los objetos nos permiten en este caso mantener organizados y agrupados los valores relativos al objeto libro.

Representar resultados

Finalmente, las expresiones también pueden mostrar los valores resultantes del calculo de una operación compleja o una función.

Cuando una función se encuentra dentro de un objeto se le llama un método, en este caso el controlador es el objeto en el que se encuentran nuestras funciones, por eso las llamaremos métodos.

Comenzaremos agregando un método simple dentro del controlador en el que hemos trabajado los ejemplos anteriores de este capítulo. Incluye este código en el controlador, después de los valores del objeto “libro”:

```
1 $scope.numeroAleatorio = function(){  
2     return Math.floor( Math.random() * 100 );  
3 }
```

Una vez más, no te preocupes por la sintaxis , la estudiaremos más tarde, por ahora enfoquémonos en que este método llamado “numeroAleatorio” devuelve el resultado de una operación que genera un número aleatorio del 1 al 100.

Representar en el HTML los valores de un método usando una expresión es ligeramente distinto a lo que hemos utilizado hasta ahora, en este caso además de usar el nombre que usamos en el controlador (sin usar el \$scope, claro) le debemos incluir paréntesis “()”, en nuestro ejemplo, deberías incluir este párrafo:

```
1 <p>Número aleatorio: {{ numeroAleatorio() }}</p>
```

Como ves, en la expresión esta incluido el nombre del método con paréntesis, eso ejecuta el proceso y muestra su resultado. Toma en cuenta que si no incluyes los paréntesis cuando invocas un método, no se mostrarán los valores correctamente.

La expresión que acabamos de incluir muestra un número diferente cada vez que refrescas el documento

Numero aleatorio: 73

El código completo de la aplicación que hemos creado, con todos los ejemplos y quedaría así:

```
1  <!DOCTYPE html>
2  <html ng-app="expresionesApp">
3  <head>
4      <meta charset="utf-8">
5      <title>expresiones</title>
6      <meta name="description" content="">
7      <meta name="viewport" content="width=device-width">
8  </head>
9  <body>
10
11      <div ng-controller="miControlador">
12          {{ mensaje }} {{ segundoMensaje }}
13
14          <p> Este libro fue escrito por {{libro.autor}}, se llama " {{libro.titulo}}
15 o}} " y habla sobre {{libro.tema}}</p>
16
17          <p>Número aleatorio: {{ numeroAleatorio() }}</p>
18      </div>
19
20      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.min.js"></script>
21
22
23      <script>
24          var aplicacionDeExpresiones = angular.module('expresionesApp', []);
25
26          aplicacionDeExpresiones.controller('miControlador', function ($scope) {
27              $scope.mensaje = "Las Expresiones AngularJS";
28              $scope.segundoMensaje = " ...te dejan agregar muchos valores!";
29
30              $scope.libro = {};
```



```
31     $scope.libro.autor = "Carlos Solis";
32     $scope.libro.titulo = "Manual del guerrero";
33     $scope.libro.tema = "AngularJS";
34     $scope.numeroAleatorio = function(){
35         return Math.floor( Math.random() * 100 );
36     }
37     });
38     </script>
39 </body>
40 </html>
```

Felicitaciones ¡Ya sabes como funcionan las expresiones de AngularJS!

En este capítulo aprendimos a entrelazarlas con el código HTML, a usar valores estáticos, dinámicos, objetos y hasta aplicar métodos con operaciones complejas.

Filtros en expresiones

En AngularJS tenemos varios tipos de filtros, para diferentes usos y contextos, los filtros de las expresiones permiten darle formato a textos o números representados dentro de una expresión.

Puedes asignarle filtros a cualquier expresión de tu aplicación, de hecho es posible utilizarlos en conjunto y hasta darles formatos específicos, los filtros de las expresiones son muy flexibles y se adaptarán a la mayoría de los posibles escenarios de trabajo.

Para insertar un filtro dentro de expresión tenemos que insertar dentro de ella el símbolo pleca “|” inmediatamente después del valor representado en la expresión. Seguido de este símbolo se incluye el filtro que se desea aplicar a esta expresión.

A manera de ejemplo general, los filtros usan este patrón:

```
1 {{ VALOR | FILTRO }}
```

AngularJS nos ofrece muchos tipos diferentes de filtros, comenzaremos estudiando los más básicos, los que podemos utilizar directamente en conjunto con una expresión

Representar Divisas

Las aplicaciones que usan comercio electrónico o muestran catálogos de productos generalmente necesitan representar los precios usando una moneda o divisa específica, por ejemplo el precio en dólares o euros usando el símbolo correspondiente (\$, €, ¥ , etc..). Sin embargo, dentro de una app las operaciones usan datos numéricos únicamente, el símbolo de las monedas, no es más que un elemento cosmético.

La forma de representar símbolos de monedas en AngularJS es a través de un filtro específico para divisas (o currency en inglés). Este filtro se utiliza para convertir un valor numérico en un formato monetario. Su uso más común va a ser para representar valores o precios de productos o servicios.

Vamos a crear un ejemplo que utilice el filtro de divisas, comenzaremos con una aplicación móvil que represente un valor numérico:

```
1 <!DOCTYPE html>
2 <html ng-app="filtros">
3   <head>
4     <meta charset="utf-8">
5     <title>Filtros de expresiones </title>
6     <meta name="viewport" content="width=device-width">
7   </head>
8   <body>
9
10    <h1>Angular Café</h1>
11
12
13    <h2>Café Americano: {{ 12 }}</h2>
14    <h2>Café Latte: {{ 12 }}</h2>
15
16
17    <script src="angular.min.js" ></script>
18    <script>
19      var filtros = angular.module('filtros', [])
20    </script>
21  </body>
22 </html>
```

Nuestro ejemplo muestra los precios de una cafetería ficticia que por ahora vende solamente 2 tipos de café: Americano y Latte



Angular Café

Café Americano: 12

Café Late: 12

Sin embargo, la aplicación tiene un problema fundamental: aunque los precios aparecen

correctamente, no sabemos en que moneda están. El usuario no puede saber si hablamos de 12 dólares americanos, 12 pesos mexicanos o 12 yenes y en cada caso el valor cambia radicalmente. Vamos a solucionar el problema agregando el filtro “currency” que le dará formato de divisa a nuestros precios, para hacerlo solo debes agregar el símbolo pleca “|” seguido del nombre del filtro. Las expresiones de nuestro ejemplo ahora deben verse así:

- 1 `<h2>Café Americano: {{ 12 | currency }}</h2>`
- 2 `<h2>Café Latte: {{ 12 | currency }}</h2>`

Solo el hecho de agregar este filtro ha cambiado notablemente nuestra aplicación, ahora los precios aparecen en dólares y con decimales, la forma en que se representan comúnmente los precios:



Angular Café

Café Americano: \$12.00

Café Late: \$12.00

Como puedes notar en este caso el valor de los precios se está representando en formato de dólares americanos. Por defecto AngularJS asignará cualquier valor en formato de divisas con la moneda norteamericana. Esto no significa que en adelante tendrás que vender todos tus productos en dólares americanos, por el contrario, AngularJS tiene soporte para cualquier moneda con solo algunos cambios básicos en la sintaxis del filtro.

Vamos a modificar la divisa de los cafés de nuestro ejemplo y definiremos dicho precio en 12 euros. Para cambiar el tipo de divisa, solo debes agregar dos puntos “:” y el símbolo de la divisa después del filtro “currency”. En nuestro caso el símbolo es “€” así que para representar el precio de los cafés en euros debemos modificar el código así:

- 1 `<h2>Café Americano: {{ 12 | currency : "€" }}</h2>`
- 2 `<h2>Café Late: {{ 12 | currency : "€" }}</h2>`



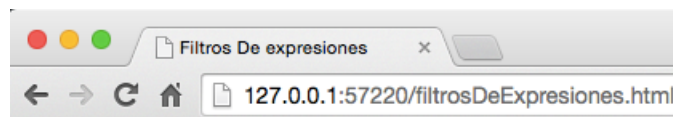
Angular Café

Café Americano: €12.00

Café Late: €12.00

El mismo principio se puede aplicar a cualquier otra moneda o texto, solo tienes que incluir el símbolo correspondiente. Por ejemplo debido a que varios países usan el símbolo “\$” para representar su moneda, si deseas identificar que los precios usan dólares americanos, puedes definirlo en el filtro mezclando texto con símbolos:

- 1 `<h2>Café Americano: {{ 12 | currency : "USD $" }}</h2>`
- 2 `<h2>Café Late: {{ 12 | currency : "USD $" }}</h2>`



Angular Café

Café Americano: USD \$12.00

Café Late: USD \$12.00

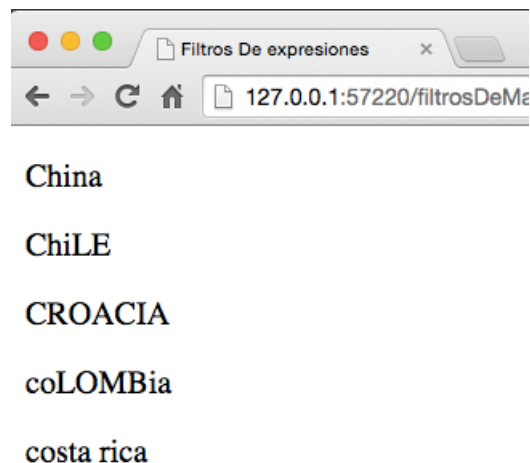
Modificando las mayúsculas

Frecuentemente, cuando se utilizan fuentes externas de datos, como webservices, XMLs o JSONs te tienes que enfrentar a datos con formatos inconsistentes mezclando mayúsculas y minúsculas.

En el siguiente ejemplo estamos representando 5 países, pero los datos tienen formato inconsistente.

```
1  <!DOCTYPE html>
2  <html ng-app="filtros">
3    <head>
4      <meta charset="utf-8">
5      <title>Filtros de expresiones </title>
6      <meta name="viewport" content="width=device-width">
7    </head>
8    <body>
9
10     <p> {{ "China" }}</p>
11     <p> {{ "ChiLE" }}</p>
12     <p> {{ "CROACIA" }}</p>
13     <p> {{ "coLOMBia" }}</p>
14     <p> {{ "costa rica" }}</p>
15
16     <script src="angular.min.js" ></script>
17     <script>
18       var filtros = angular.module('filtros', [])
19     </script>
20   </body>
21 </html>
```

Representar valores así, aunque provengan de documentos externos, hace que tu sitio se vea poco profesional

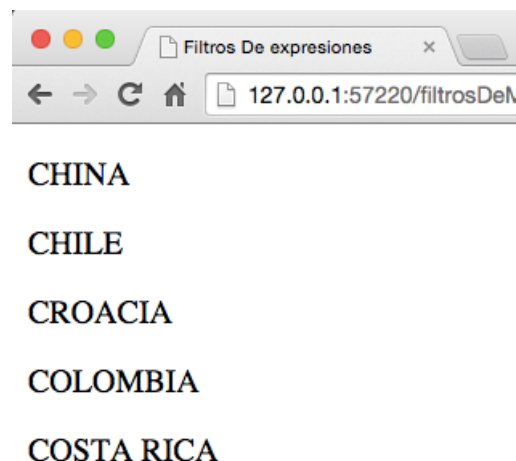


¡No temas, AngularJS también tiene una solución para esto!

De hecho, tiene dos filtros diferentes para ayudarte en estos casos. El primer filtro convierte todo el texto a mayúsculas, para aplicarlo, tienes que insertar el filtro “uppercase” después del respectivo símbolo “|” que hemos utilizado anteriormente. En nuestro ejemplo, para que todos los países aparezcan en mayúsculas deberías modificar tu código así:

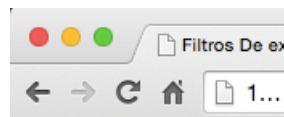
```
1 <p> {{ "China" | uppercase }}</p>
2 <p> {{ "ChiLE" | uppercase }}</p>
3 <p> {{ "CROACIA" | uppercase }}</p>
4 <p> {{ "coLOMBia" | uppercase }}</p>
5 <p> {{ "costa rica" | uppercase }}</p>
```

Aunque el valor original de la expresión sigue con un formato inconsistente, el filtro “uppercase” los mostrará a todos en un formato de mayúsculas



Si por el contrario quieres modificar el formato del texto a minúsculas, el procedimiento es incluir el filtro “lowercase”

```
1 <p> {{ "China" | lowercase }}</p>
2 <p> {{ "ChiLE" | lowercase }}</p>
3 <p> {{ "CROACIA" | lowercase }}</p>
4 <p> {{ "coLOMBia" | lowercase }}</p>
5 <p> {{ "costa rica" | lowercase }}</p>
```



china

chile

croacia

colombia

costa rica

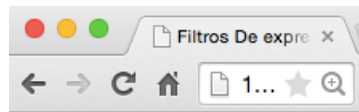
Hasta aqui trabajan los filtros de AngularJS pero si tu eres como yo supongo que ya te estarás preguntando ¿Como se muestra la primera letra en mayúscula? En este caso, no hay un filtro específico para esta conducta, pero puedes lograrla con solo una línea de código en CSS usando la propiedad:

```
1 text-transform: capitalize;
```

Aplicado a nuestro ejemplo, solo tienes que incluir este código antes de la etiqueta </head>

```
1 <style>
2   p{
3     text-transform: capitalize;
4   }
5 </style>
```

El resultado, es una lista con un formato consistente y una apariencia mucho más profesional que la versión con la que arrancamos el ejemplo.. ¡Misión Cumplida AngularJS!



China

Chile

Croacia

Colombia

Costa Rica

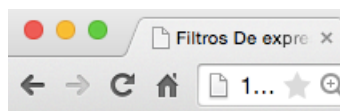
Representando números

Los números son uno de los tipos de datos que más frecuentemente encontraras en una expresión. Al inicio del capítulo aprendimos a darle un formato de divisa a los números, pero aún nos quedan algunos trucos bajo la manga para representarlos.

El filtro “number” puede forzar cadenas de texto o booleanos a que se comporten como números, por ejemplo la siguiente expresión contiene uno:

```
1 <p> {{ false }}</p>
```

El valor que representa esta expresión en el navegador es “False”

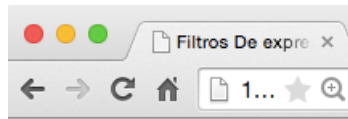


False

Al utilizar el filtro “number” convertimos el booleano a un valor numérico:

```
1 <p> {{ false | number }}</p>
```

Los booleanos solo pueden contener dos valores: 0 para false y 1 para true, así que en este caso el valor de nuestra expresión será de cero.

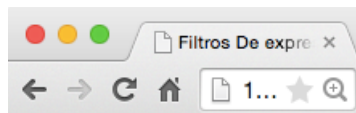


0

Este filtro también puede manipular los decimales de un número, por ejemplo, supongamos que recibes el valor de Pi (π) con 12 decimales y solo deseas que se muestre con 2 de ellos, puedes limitar esta cifra con el filtro “number” agregando dos puntos “:” y la cantidad de decimales deseados, en el caso que mencionamos la expresión sería similar a esta:

```
1 <p> {{ 3.141592653589 | number: 2 }} </p>
```

Gracias al control que efectúa el filtro “number” aunque el valor es de 12 dígitos, solo se muestran dos en tu documento:

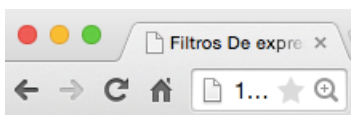


3.14

Aplicando la misma lógica, puedes convertir un número decimal a un número entero reduciendo a cero la cantidad de decimales que se muestran

```
1 <p> {{ 15.26535899744 | number: 0 }} </p>
```

El filtro “number” modifico 15.26535899744 al entero 15

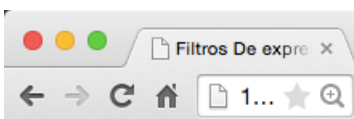


15

Estos filtros para números, pueden ser particularmente útiles para mostrar precios, de hecho podemos combinar el filtro de divisas con el de números, en este ejemplo tenemos un postre con un valor definido en 6 decimales, pero usaremos una combinación de filtros para mostrarlos en un formato más amigable:

```
1 <p> {{ 22.205309 | number: 2 | currency: "MXP $" }} </p>
```

El filtro que usamos reduce a solo dos dígitos la representación del precio y además lo despliega en formato de pesos mexicanos, todo en una sola línea de código.



MXP \$22.21

Formato de Fechas

Las fechas son útiles no solo para informar al usuario, también son cruciales para la logística, administración y coordinación de un sinfín de aplicaciones, desde tickets de aerolíneas hasta tiendas con productos perecederos.

Es muy común que la información de fechas nos llegue desde el servidor en formatos ilegibles para una persona. Usualmente cuando las fechas vienen en formatos difíciles de leer se convierten a cadenas de texto más legibles, esta técnica, aunque ayuda a que el usuario comprenda la información puede generar otro problema: si el usuario cambia ese valor es necesario volver a convertirlo al formato original.

AngularJS tiene soporte para leer la mayoría de los formatos de fechas y puede adaptarlos para que sean fáciles de leer sin alterar sus valores, ahorrando tiempo y cálculos innecesarios.

En este ejemplo tenemos 2 formatos diferentes para representar una fecha: El primero es el formato UNIX EPOCH, una referencia temporal usada en muchos lenguajes de programación tales como JavaScript, PHP, o Ruby entre otros; El segundo formato utiliza el standard ISO-8601 soportado también por la mayoría de lenguajes de programación modernos.

```
1 946641599000
2 1999-12-31T23:59:59+00:00
```

En ambos casos la fecha representada es 31 de diciembre de 1999, un segundo antes del cambio de milenio. La forma más simple de darle formato a esta fecha es incluyendo el filtro “date”

```
1 {{ '946641599000' | date }}
```

El resultado es un valor de texto que muestra el año, día y mes indicado



El formato utilizado por defecto esta al estilo usado en los Estados Unidos (MES/DIA/AÑO) pero AngularJS tiene soporte para personalizar el formato, por ejemplo si queremos que se use el estilo de fecha en español mostrando primero el día, mes y año, el código debería ser:

```
1 {{ '946684799000' | date: 'dd/MM/yy' }}
```



Por último, también puedes usar formatos predefinidos como “short” o “mediumDate” que muestran la fecha usando formas comunes:

```
1 {{ '946684799000' | date:'mediumDate' }}
```



Puedes encontrar una lista completa con los parámetros de fechas en la documentación oficial de AngularJS

<https://docs.angularjs.org/api/ng/filter/date>

Encadenar filtros

Tal como lo vimos en ejemplos anteriores, los filtros de AngularJS pueden trabajar en conjunto, este concepto se llama encadenamiento de filtros y a manera de ejemplo general funciona agregando una pleca “|” como separador entre filtros. La estructura de una expresión con filtros múltiples sería así

```
1 {{ EXPRESION | FILTRO1 | FILTRO2 }}
```

Puedes incluir todos los filtros que desees en el orden que quieras, todos ellos se aplicarán sobre la expresión y mostrarán un resultado combinado en tu documento, por ejemplo, si quieres presentar una fecha con caracteres en mayúsculas, puedes usar “date” junto a “uppercase”

```
1 {{ '1405222862000' | date:'fullDate' | uppercase }}
```

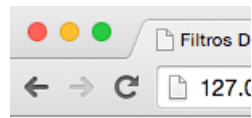
En esta caso, la expresión es un dígito en milisegundos al que se le aplica formato de fecha y se modifican todos sus caracteres alfabéticos a mayúsculas.



Todos los filtros pueden encadenarse sin afectar el resultado, en este otro ejemplo tomaremos una expresión que muestra una cadena de texto, convertiremos todo a minúsculas y aplicaremos el filtro “limitTo” que permite restringir la cantidad de caracteres desplegados.

```
1 {{ "Manual del Guerrero: AngularJS" | lowercase | limitTo:6 }}
```

El resultado es que sin modificar el valor original de la expresión, estamos mostrando únicamente 6 caracteres en minúsculas



manual

Filtros personalizados

Hasta ahora, hemos explorado los principales y más comunes filtros de AngularJS, sin embargo, es muy posible que cuando estés trabajando con una aplicación compleja o de gran escala, la funcionalidad de los filtros que estudiamos se quede corta para tus necesidades.

Afortunadamente en AngularJS puedes crear tus propios filtros para que se ajusten a la medida a cualquier requerimiento que tenga tu aplicación. Los filtros personalizados una vez creados se pueden incorporar en una aplicación con el mismo formato y propiedades que cualquier otro filtro.

Si deseas crear un filtro personalizado, primero necesitas tener una aplicación AngularJS debidamente instalada y configurada. Vamos a crear un nuevo filtro así que usaremos este código con una aplicación AngularJS lista para comenzar a trabajar.

```
1  <!DOCTYPE html>
2  <html ng-app="filtros">
3      <head>
4          <meta charset="utf-8">
5          <title>Filtros de expresiones </title>
6          <meta name="viewport" content="width=device-width">
7
8      </head>
9      <body>
10
11          <h1> </h1>
12
13          <script src="angular.min.js" ></script>
14
15          <script>
16
17              var filtroApp = angular.module('filtros', [])
18
19          </script>
20      </body>
21  </html>
```

Quiero que pongas especial atención a que “filtroApp” es el nombre de la variable con la que hemos declarado la aplicación de este documento, a partir de este punto, para agregar elementos vamos a utilizar esta variable.

Comenzaremos primero agregando una expresión dentro de la etiqueta <h1>, para este ejemplo usare una cadena de texto con la palabra “odontologo”, para efectos prácticos voy a omitir intencionalmente la tilde.

```
1 <h1>{{ "odontologo"}}</h1>
```

Procederemos ahora a declarar el filtro personalizado, en este ejemplo ubicaremos este código en la línea inmediatamente posterior a la declaración de la aplicación:

```
1 filtroApp.filter('cambiaVocal', function () {
2
3     var cambiaVocalFiltro = function (datosOriginales, argumento) {
4
5         return "resultado";
6     };
7
8     return cambiaVocalFiltro;
9
10 });
```

Este es el código base para registrar un nuevo filtro en AngularJS, antes de avanzar sobre este tema, voy a explicarte línea por línea, lo que acabamos de escribir. Comenzamos por la primera línea:

```
1 filtroApp.filter('cambiaVocal', function () {
```

Aquí estamos utilizando “filtroApp” como una instancia de nuestra aplicación y cada cosa que agreguemos allí será incluido en la aplicación. Por esa razón se incluye el método “filter” que nos permite crear un nuevo filtro en AngularJS, a ese filtro que estamos creando le definimos dos parámetros, el primero es el nombre con el cual lo identificaremos en el sistema, en este caso, será “cambiaVocal”, el segundo parámetro es una función, es allí donde incluiremos el código que compone el filtro.

```
1 var cambiaVocalFiltro = function (datosOriginales, argumento) {
2     return "resultado";
3 };
```

En esta línea comenzamos a definir la conducta propia de nuestro filtro, estamos declarando una variable llamada “cambiaVocalFiltro” donde reside otra función. En este caso la función tiene 2 argumentos, notarás que los nombres designados son “datosOriginales” y “argumento” para que te sea más fácil comprender su uso. Sin importar los nombres que le asignes a estos

argumentos, el primero siempre recibe los datos originales que se encuentran en la expresión y el segundo recibe los argumentos o valores opcionales del filtro.

Dentro de esta función también encontraras un “return” que es el comando de JavaScript para finalizar esta función y devolver un resultado, por ahora solo hay una cadena de texto estática con el valor “resultado”, este será el valor que se muestre cada vez que se aplique el filtro que creamos.

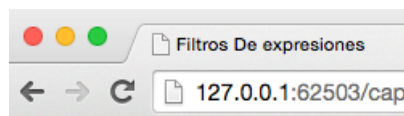
```
1 return cambiaVocalFiltro;
```

La ultima parte de nuestro filtro es un “return” que devuelve el resultado de la función “cambiarVocalFiltro” que creamos antes. La arquitectura de AngularJS requiere que los filtros sigan este patrón devolviendo como resultado una función que puede recibir y procesar los valores de la expresión.

Técnicamente ya esta listo nuestro filtro, solo necesitas invocarlo como cualquier otro filtro dentro de la expresión del documento:

```
1 <h1>{{ "odontologo" | cambiaVocal }}</h1>
```

Al probar este documento en el navegador obtendrás el resultado que devuelve la función “cambiaVocalFiltro”



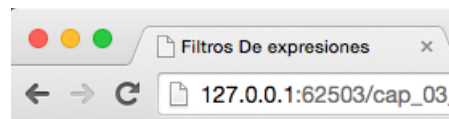
resultado

Por ahora, este filtro aunque hace lo que le pedimos, es bastante limitado... ¡Es hora de utilizar valores dinámicos!

Supongamos que el objetivo del filtro que hemos creado es reemplazar las vocales de la expresión, para que esto ocurra vamos a reemplazar el código actual de “cambiaVocalFiltro” por este:

```
1  var cambiaVocalFiltro = function (datosOriginales, argumento) {  
2  
3      var nuevosDatos = datosOriginales.replace(RegExp("o", "g"), "_")  
4  
5      return nuevosDatos;  
6  };
```

Ahora la función tiene una nueva variable que toma los valores de “datosOriginales”, o sea, los datos que tiene actualmente la expresión y les aplica una expresión regular de JavaScript que reemplaza cada ocurrencia de la letra “o” por el caracter “_”.



_d_nt_l_g_

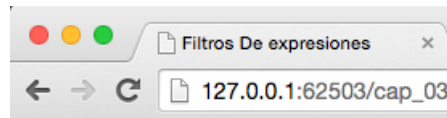
Las cosas comienzan a ponerse entretenidas, pero podemos seguir mejorando este filtro, por ejemplo puedes usar argumentos para definir dinámicamente que caracter usar para reemplazar la vocal “o”. Para realizar este proceso, primero agregamos un argumento en el filtro:

```
1  <h1>{{ "odontologo" | cambiaVocal:"*" }}</h1>
```

Ahora tenemos que modificar el filtro para usar el nuevo argumento. Como notarás en la variable “cambiaVocalFiltro” tenemos una función con 2 argumentos, el primero ya vimos que recibe el valor original de la expresión, el segundo por otro lado es el que recibe el valor del argumento que acabamos de definir. Lo que nos toca ahora es incluir este valor en el comando que modifica el texto. Reemplaza la variable “nuevosDatos” por este código:

```
1  var nuevosDatos = datosOriginales.replace(/o/g, argumento)
```

Ahora nuestro filtro usa el valor del argumento para reemplazar la letra “o”



d*nt*l*g

También, puedes hacer filtros con soporte para múltiples argumentos. Vamos a aplicar este concepto en nuestro ejemplo modificando la expresión:

```
1 <h1>{{ "odontologo" | cambiaVocal:"*":"o"}}</h1>
```

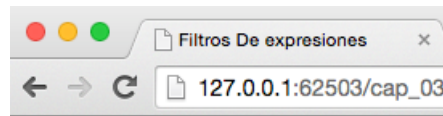
Ahora, el filtro “cambiaVocal” tiene dos argumentos. Debemos entonces modificar el código del filtro para poder procesar este nuevo valor, solo es necesario agregar un tercer argumento en la función “cambiaVocalFiltro” :

```
1 var cambiaVocalFiltro = function (datosOriginales, argumento , segundoArgumento)\
2 {
3
4     var nuevosDatos = datosOriginales.replace(RegExp(segundoArgumento, "g"), argu\
5 mento)
6
7     return nuevosDatos;
8 };
```

También, como puedes notar, he modificado los valores de “nuevosDatos” para que la expresión regular use el nuevo valor de “segundoArgumento”. En este punto el filtro sigue mostrando el mismo resultado, pero gracias a que los valores son dinámicos, podemos modificar su comportamiento, por ejemplo aquí he cambiado mi filtro para cambiar la letra “a” por un guión “-”

```
1 <h1>{{ "abracadabra" | cambiaVocal:"-":"a"}}</h1>
```

Puedes probar con tus propias combinaciones de letras y reemplazos, el filtro debería manejarlo sin problema



-br-c-d-br-

Como puedes ver ¡El código en AngularJS tiene su magia!

Filtros de Datos

Los filtros básicos de AngularJS tienen una función principalmente enfocada al formato, sin embargo su funcionalidad se ve limitada en el momento de manipular datos complejos.

Cuando tenemos una gran cantidad de datos es muy posible que necesitemos agruparlos o filtrarlos para analizar únicamente los que necesitamos, aquí es donde entran en acción los filtros de datos de AngularJS.

Filtrando datos

AngularJS hace muy sencillo en proceso de búsqueda y filtrado de datos. Vamos a poner a trabajar este tipo de filtros y crearemos una aplicación que nos permitirá buscar datos en tiempo real.

Comenzaremos creando una aplicación con un controlador y dentro de éste, incluiremos un arreglo con los datos que vamos a filtrar.

```
1  <!DOCTYPE html>
2  <html ng-app="filtrarlistas">
3
4  <head>
5      <meta charset="utf-8">
6      <title>Filtrando Datos</title>
7      <meta name="viewport" content="width=device-width">
8  </head>
9
10 <body>
11
12     <div ng-controller="listas">
13
14         <ul>
15             <li ng-repeat="fruta in frutas">{{fruta}}</li>
16         </ul>
17     </div>
18
19     <script src="angular.min.js">
20 </script>
21
```

```
22     <script>
23         var miApp = angular.module('filtrarlistas', [])
24
25         miApp.controller('listas', function ($scope) {
26             $scope.frutas = ['Kiwi', 'Banana', 'Arándano']
27         })
28     </script>
29
30 </body>
31
32 </html>
```

Esta aplicación usa la directiva “ng-repeat” para representar los datos del arreglo en forma de lista:

- Kiwi
- Banana
- Arándano

Vamos a usar AngularJS para crear un buscador en tiempo récord ¡Comienza a contar los segundos!

Comenzaremos creando la casilla de texto donde realizaremos la búsqueda, agrega este código justo antes de la actual lista

```
1 <input type="text" ng-model="buscaFruta">
```

Este código no solo incluye una entrada de texto, también utiliza la directiva “ng-model” para crear un nuevo valor dentro de la aplicación llamado “buscaFruta” ahora los contenidos de la casilla de texto están vinculados a dicho valor.

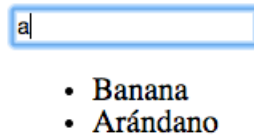
Para que nuestro buscador comience a funcionar, solo es necesario agregar el nuevo filtro enlazando los valores de esta búsqueda, recordemos que los filtros siempre vienen precedidos de una pleca “[” y el nombre del filtro, en este caso usaremos uno llamado justamente “filter”. La función de éste es mostrar únicamente los valores de un conjunto que calcen con sus parámetros de búsqueda.

En este caso, queremos filtrar los valores de la lista de acuerdo a “buscaFruta” así que debemos modificar la directiva “ng-repeat” de esta forma:

```
1 <ul>
2   <li ng-repeat="fruta in frutas | filter:buscaFruta">{{fruta}}</li>
3 </ul>
```

¡Para de contar! Eso fue todo ¡Ya tienes un buscador funcional!

Guarda el nuevo código y ábrelo en tu navegador para que lo veas en acción.



De acuerdo al código que acabas de agregar, solo se mostrarán en la lista los elementos que tengan caracteres en común con el valor existente en “buscaFruta”, como este valor se relaciona a la casilla de texto, cada vez que escribas un texto en la casilla, la lista puede modificarse.

Por ejemplo, si en la casilla de texto solo escribes la letra “a”, notarás que solo aparecen los ítems: “Banana” y “Arándano” porque contienen esa letra, mientras que si escribes solo la letra “k”, la lista solo mostrara “Kiwi” por ser el único ítem que calza con ese parámetro de búsqueda.

Ordenar colecciones de datos

AngularJS no solo nos permite hacer búsquedas sobre arreglos simples, también podemos trabajar con objetos más complejos.

Continuaremos usando el ejemplo anterior pero esta vez vamos a modificar el valor de `$scope.frutas` por un objeto que contenga varias propiedades por cada fruta:

```
1 $scope.frutas = [
2   {
3     nombre: 'Kiwi',
4     color: 'Verde',
5     propiedades: 'rica en vitamina C'
6   },
7   {
8     nombre: 'Banana',
9     color: 'Amarilla',
10    propiedades: 'rica en magnesio y potasio'
11  },
```



```
12     {
13         nombre: 'Arándano',
14         color: 'Azul',
15         propiedades: 'buen Antioxidante'
16     },
17 ]
```

Ahora, tenemos un arreglo más complejo, en vez de almacenar cadenas de texto simples, almacena 3 objetos, cada uno de ellos a su vez contiene 3 propiedades con valores individuales.

Procedemos a modificar también la lista para que muestre correctamente estos nuevos valores, el nuevo código de la lista será así:

```
1 <ul>
2   <li ng-repeat="fruta in frutas | filter:buscaFruta">
3     <h2>{{fruta.nombre}}</h2>
4     <p>Fruta de color {{fruta.color}} conocida por ser {{fruta.propiedades}}\
5   </p>
6   </li>
7 </ul>
```

Aunque usa la misma sintaxis para crear la lista, está mostrando la información usando las propiedades de cada objeto.

• Kiwi

Fruta de color Verde conocida por ser rica en vitamina C

• Banana

Fruta de color Amarilla conocida por ser rica en magnesio y potasio

• Arandano

Fruta de color Azul conocida por ser buen Antioxidante

Como notarás, el orden de los elementos, es el mismo que tiene en la lista original: Kiwi es el primero y Arándano es el ultimo, utilizaremos un filtro más de AngularJS para ordenar estos elementos por orden alfabético

El filtro que usaremos para ordenar los elementos de la lista se llama “orderBy” .Al igual que cualquier otro filtro lo incluiremos precedido de una pleca, pero esta vez usaremos una sintaxis especial en sus valores. Ya que estamos trabajando con un objeto, no basta con definir este filtro, también hay que especificar bajo que valor debe realizarse el orden, por ejemplo para ordenar los elementos a partir de los contenidos de la propiedad “nombre”,agregaremos este filtro a la lista de nuestro ejemplo:

```
1 | orderBy: '+nombre'
```

Si pones atención, además de poner el nombre de la propiedad entre comillas, también estamos usando el signo “+” que ordena la lista en orden regular, el resultado ya muestra los elementos ordenados según su nombre

- **Arándano**

Fruta de color Azul conocida por ser buen antioxidante

- **Banana**

Fruta de color Amarilla conocida por ser rica en magnesio y potasio

- **Kiwi**

Fruta de color Verde conocida por ser rica en vitamina C

Si cambias el signo por “-” los elementos serán desplegados en orden inverso

```
1 | orderBy: '-nombre'
```

- **Kiwi**

Fruta de color Verde conocida por ser rica en vitamina C

- **Banana**

Fruta de color Amarilla conocida por ser rica en magnesio y potasio

- **Arándano**

Fruta de color Azul conocida por ser buen antioxidante

Al usar en conjunto todos los filtros tenemos una lista que no solo se muestra en orden alfabético, también puede realizar búsquedas sobre las propiedades y mantener el orden definido.

Toma en cuenta que el comportamiento de la búsqueda cuando trabajas con un objeto de varias propiedades es analizar la cadena de texto solicitada sobre TODAS las propiedades de TODOS los objetos.

En nuestro ejemplo puedes buscar la cadena “rica en” que calza con el contenido del valor propiedades en dos diferentes registros, al mostrar los resultados se mantendrá también el orden alfabético.

- **Kiwi**

Fruta de color Verde conocida por ser rica en vitamina C

- **Banana**

Fruta de color Amarilla conocida por ser rica en magnesio y potasio

Controladores

Los controladores son una parte integral de AngularJS y son los bloques fundamentales para crear cálculos y procesos en una aplicación.

Gracias a la arquitectura separada en capas de Angular, podemos administrar por completo una aplicación desde sus controladores que a su vez mantienen un principio de encapsulamiento el cual permite que cada controlador se maneje individualmente sin afectar los datos o procesos de los demás controladores con los que pueda coexistir.

Los controladores de AngularJS tienen la capacidad de administrar datos, realizar cálculos y modificar el DOM de un documento y serán la principal herramienta que usaremos para interactuar con las acciones de nuestros usuarios.

Creando un controlador

Un controlador solo puede crearse dentro de un documento debidamente configurado para ejecutar AngularJS y está relacionado directamente con la aplicación que administra el documento.

Vamos a crear un controlador y comenzaremos con un código base que incluya una aplicación AngularJS:

```
1  <!DOCTYPE html>
2  <html ng-app="controladores">
3
4  <head>
5      <meta charset="utf-8">
6      <title>Controladores</title>
7      <meta name="viewport" content="width=device-width">
8  </head>
9
10 <body>
11
12     <div>
13     </div>
14
15
16     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.m\
17 in.js"></script>
```

```
18
19     <script>
20         var controladorApp = angular.module('controladores', [])
21
22     </script>
23 </body>
24
25 </html>
```

En este documento usamos la directiva “ng-app” para nombrar nuestra aplicación “controladores” y la variable “controladorApp” sirve para declararla, en este punto ya tenemos un documento listo para trabajar.

Vamos a crear un controlador agregando este código en la línea inmediatamente posterior de que declaramos la aplicación con la variable “controladorApp”:

```
1 controladorApp.controller('miPrimerControlador' , function($scope){
2
3 })
```

Gracias a la variable que declaré antes del controlador , puedo usar “controladorApp” como una instancia para manipular mi aplicación, en este caso el método “controller” me permite registrar nuevos controladores.

Observa como “controladorApp.controller” recibe dos valores: El primero será el nombre del controlador, en este caso “miPrimerControlador”, el segundo es una función que contiene los comando e instrucciones del mismo.

Esta función, que es el corazón de mi controlador, utiliza como argumento un objeto muy particular de AngularJS: **\$scope**.

Hablaremos de este objeto en un momento, por ahora ¡Vamos a terminar de instalar el controlador!

Activar el controlador en un documento

Aunque para este punto tu controlador esta efectivamente registrado en la aplicación, aun no tiene ningún uso o efecto en tu documento.

En AngularJS los controladores se activan utilizando la directiva “ng-controller”. Continuando con el ejemplo que estamos trabajando vamos a agregar el controlador que acabamos de crear para que controle una parte de nuestro documento, reemplazaremos el <div> que se encontraba por este nuevo código:

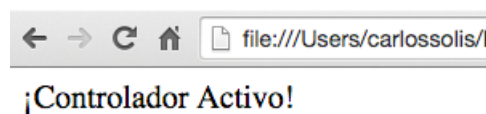
```
1 <div ng-controller="miPrimerControlador">
2   {{ mensaje }}
3 </div>
```

En este caso hemos usado la directiva “ng-controller” para registrar el controlador “miPrimerControlador”. También he aprovechado para incluir una expresión que representa una propiedad llamada “mensaje”. Si ejecutas tu aplicación en este momento obtendrás una pantalla en blanco porque esa propiedad aun no tiene ningún valor.

Modificaremos ahora el controlador, ahora que esta relacionado directamente a ese <div> le asignaremos un valor a “mensaje”:

```
1 controladorApp.controller('miPrimerControlador' , function($scope){
2
3     $scope.mensaje = "¡Controlador Activo!"
4
5 })
```

Ahora si puedes guardar tu documento y ejecutarlo, obtendrás el mensaje del controlador dentro del <div> que estaba vacío.



Estamos usando una vez más el objeto \$scope esta vez para definir un valor, vamos a explorar en detalle este objeto.

El objeto \$scope

Al crear un controlador no solo estamos haciendo una simple función, sino una conexión entre AngularJS y el documento en el que se ejecuta. Tu controlador podrá administrar eventos, modificar valores y hasta cambiar la apariencia del documento, para todo esto necesitas usar a \$scope.

Recuerda que los controladores están encapsulados, todos sus cálculos y valores existen únicamente dentro de si mismos, el objeto \$scope es el puente entre el controlador y el documento lo que permite enviar y recibir valores, propiedades y eventos entre ambas partes.

Para que te imagines mejor como funciona el objeto \$scope, te doy un ejemplo del mundo cotidiano: ¿Recuerdas el bolso de Mary Poppins? Ese al que le cabía de todo, desde una

lámpara hasta una florero, bueno, imagina que al usar un controlador en una etiqueta, metes allí todo lo que está en ella, no importa la cantidad de elementos que tenga, todo lo que entra en el bolso está ahora relacionado entre si y separado de los elementos externos. Cuando necesitas modificar algo, se lo pides a la Señorita Poppins. Nadie sabe como diablos ella se las arregla, pero solo necesita meter la mano en su bolso y, sin siquiera mirar, te devolverá lo que buscas; ella es el nexo entre los ítems del bolso y el exterior, el objeto `$scope` trabaja de la misma forma ¡pero sin necesidad de que hagas ningún número musical!

Siempre que necesites crear un valor que tenga efecto dentro del documento debes hacerlo a través de `$scope`, vamos a modificar ligeramente el controlador para explicar mejor este punto

```
1  controladorApp.controller('miPrimerControlador' , function($scope){
2
3      var saludo = "¡Hola! "
4      $scope.mensaje = "¡Controlador Activo!"
5
6
7  })
```

En este controlador tenemos dos valores, el primero es una variable regular llamada “saludo” , el segundo valor se encuentra en una propiedad “mensaje” dentro de `$scope` lo que significa que esta conectada con el documento.

Veamos ahora como se comportan estos valores en tu documento agregando este código que intenta invocar ambos:

```
1  <div ng-controller="miPrimerControlador">
2      <p> - {{ mensaje }} </p>
3      <p> - {{ saludo }} </p>
4  </div>
```

La primera expresión muestra el valor de “mensaje” al cual tenemos acceso por estar dentro de `$scope`.

La segunda expresión invoca a “saludo” que aunque efectivamente es parte del controlador no tiene acceso al documento y se encuentra encapsulada dentro del controlador.



Es importante mencionar que `$scope` se utiliza únicamente dentro del controlador, nunca en una expresión. Cualquier elemento que se encuentre dentro de `$scope` en un controlador, se debe invocar directamente en el documento. Por ejemplo, aunque “mensaje” es parte de `$scope`, esta es una forma **incorrecta** de invocarlo

```
1 <div ng-controller="miPrimerControlador">
2   <p> - {{ $scope.mensaje }} </p>
3   <p> - {{ $scope.saludo }} </p>
4 </div>
```

Al utilizar una sintaxis como esa, tendrás un error en los valores y no se desplegarán correctamente, esto se debe a dentro del documento no existe el objeto `$scope` y por ende no puedes llamarlo dentro de una expresión.

Alcance de un controlador

Los controladores trabajan en áreas delimitadas de tu aplicación, eso significa que debes relacionar los controladores a una zona específica de tu documento y solo allí tendrá efecto, en el ejemplo que hemos trabajado, nuestro controlador está restringido al `<div>` en el que fue registrado

```
1 <div ng-controller="miPrimerControlador">
2 </div>
```

Puedes usar cualquier etiqueta para registrar un controlador, pero debes tomar en cuenta que solo tendrá efecto dentro de ella. Veamos un ejemplo práctico del alcance de un controlador creando uno nuevo, agrega este código en el documento que estamos usando de ejemplo, inmediatamente después de donde acaba el controlador anterior:


```
1 controladorApp.controller("alcance", function($scope){  
2  
3     $scope.color = "verde"  
4  
5 })
```

Este nuevo controlador tiene la propiedad “color” debidamente registrada en el \$scope lo que significa que podemos representarla en un documento. Vamos ahora a agregar 2 nuevos <div> al documento:

```
1 <div ng-controller="alcance">  
2     {{color}}  
3 </div>  
4  
5 <div>  
6     {{color}}  
7 </div>
```

En este código tenemos 2 etiquetas <div> separadas, ambas tienen una expresión invocando la propiedad color que acabamos de crear. La única diferencia que tienen es que el primer <div> esta relacionado con nuestro controlador a través de la directiva “ng-controller”, sin embargo, al probar este código el resultado solo muestra uno de los valores:

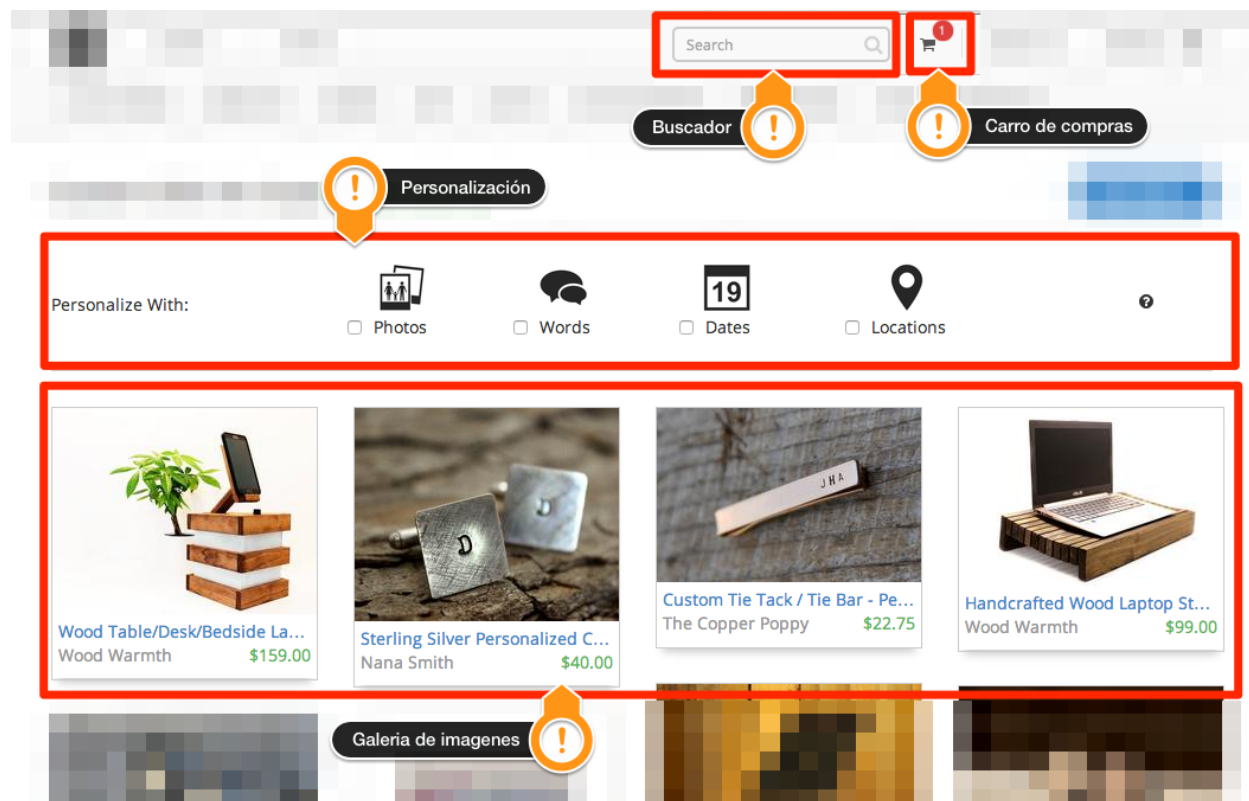


Esto se debe a que aunque la propiedad “color” esta debidamente registrada en \$scope y en ambos <div> se usa la sintaxis correcta en las expresiones, el controlador solo tiene acceso a la etiqueta en que se encuentra registrado, el resto del documento sencillamente no existe para nuestro controlador.

Mantener los controladores encapsulados es una excelente práctica de AngularJS porque nos permite mantener encapsulados los valores dentro de cada controlador, sin que afecten más elementos de los que tu definas, por eso una aplicación puede usar una cantidad indefinida de controladores de manera eficiente y ordenada.

Controladores múltiples

Gracias a que cada controlador se maneja individualmente, una aplicación AngularJS puede utilizar muchos controladores al mismo tiempo. Esto nos permite modularizar nuestras aplicaciones asignando controladores para que administren diferentes secciones y funcionalidad de un documento, pensemos por ejemplo en un sitio de comercio electrónico donde es necesario tener muchas funciones en un solo documento



En esta imagen de un sitio de ese estilo tenemos al menos 4 diferentes módulos trabajando: un buscador, un carro de compras, un módulo de personalización y una galería de imágenes con información de los productos.

Vamos a crear un nuevo documento para aplicar este concepto, utilizaremos como base este código

```
1  <!DOCTYPE html>
2  <html ng-app="controladores">
3
4  <head>
5      <meta charset="utf-8">
6      <title>Controladores</title>
7      <meta name="viewport" content="width=device-width">
8
9  </head>
10
11 <body>
12
13     <div ng-controller="frutas">
14         {{ nombre }}
15     </div>
16
17     <div ng-controller="personas">
18         {{ nombre }}
19     </div>
20
21
22     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.m\
23 in.js"></script>
24
25     <script>
26         var controladorApp = angular.module('controladores', [])
27
28
29
30     </script>
31 </body>
32
33 </html>
```

Acá ya tenemos una aplicación AngularJS configurada con un par de <div> de contenido. Observa como ambos están utilizando la misma expresión representando el valor “nombre”

```
1  {{ nombre }}
```

La diferencia entre ambos elementos es que cada uno de ellos esta relacionado a un controlador distinto: el primero esta conectado con el controlador “frutas” y el segundo con “personas”. Vamos a incluir los controles correspondientes en nuestra aplicación:

```
1  controladorApp.controller('frutas', function ($scope) {  
2  
3      $scope.nombre = "Mango"  
4  
5  })  
6  
7  controladorApp.controller("personas", function($scope){  
8  
9      $scope.nombre = "Walter Avila"  
10  
11 })
```

En estos dos controladores estamos usando una variable que se llama exactamente igual: “nombre” y esta relacionada de la misma forma al objeto \$scope pero con distintos valores.

Gracias a que la arquitectura de AngularJS restringe cada controlador a un área de contexto, los valores permanecen intactos y no generan ningún conflicto



En este caso aunque ambos usaron propiedades iguales, el objeto \$scope se enlaza únicamente con el contenido de la etiqueta correspondiente, por esta razón se representan valores diferentes.

Métodos

Hasta ahora hemos utilizado los controladores para asignar y representar valores en una aplicación, aunque imagino que vas a necesitar realizar este proceso en tus apps, estoy seguro que no estas leyendo este libro para mostrar simples datos en un documento.

Las aplicaciones AngularJS necesitan funcionalidad, realizar procesos y cálculos, no vale la pena siquiera instalar el framework para mostrar datos simples. Aquí es donde entran en la escena los métodos: las funciones que se ejecutan dentro de un controlador.

Los métodos de un controlador, al igual que las propiedades, deben crearse dentro del objeto \$scope para que puedan interactuar con el documento. Es importante mencionar que puedes

efectivamente crear métodos sin usar `$scope` dentro de un controlador, pero solo estarán disponibles para miembros del mismo controlador, piensa en este tipo de funciones como métodos privados.

Pondremos en práctica el concepto de los métodos creando una calculadora básica. Comenzaremos este nuevo ejemplo usando este código como base:

```
1  <!DOCTYPE html>
2  <html ng-app="controladores">
3
4  <head>
5      <meta charset="utf-8">
6      <title>Controladores</title>
7      <meta name="viewport" content="width=device-width">
8      <style> input, hr, button { display: block; margin: 10px 0}</style>
9  </head>
10
11 <body>
12
13     <div ng-controller="controladorConMetodos">
14
15         <input type="number" ng-model="valor1" />
16         <input type="number" ng-model="valor2" />
17
18         <button> Sumar </button>
19         <button> Multiplicar </button>
20
21         <hr>
22
23         <p>
24             {{resultado}}
25         </p>
26
27     </div>
28
29
30
31
32     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.m\
33 in.js"></script>
34
35     <script>
36         var controladorApp = angular.module('controladores', [])
```

```
37
38
39     controladorApp.controller("controladorConMetodos", function($scope){
40
41
42
43     })
44
45 </script>
46 </body>
47
48 </html>
```

Aquí ya tenemos instalado Angular y hemos declarado una aplicación que ya tiene un controlador llamado “controladorConMetodos”.

Este mismo controlador ya está enlazado al documento html a través de la directiva “ng-controller” que nos permitirá controlar todos los elementos que se encuentran en ese <div>.



Puedes observar que en documento tenemos un par de entradas de texto y dos botones que por ahora no hacen nada. Revisemos en detalle esas dos casillas de texto:

```
1 <input type="number" ng-model="valor1" />
2 <input type="number" ng-model="valor2" />
```

Como puedes notar, estas casillas de texto están usando una directiva llamada “ng-model”. Esta directiva convierte ambas casillas de texto en un “modelo” y enlaza los valores que se escriban en ellas a las variables “valor1” y “valor2”.

Más abajo en el mismo documento, encontraras una expresión “resultado”

```
1 <p>
2   {{resultado}}
3 </p>
```

Por ahora no muestra ningún dato, pero allí vamos a mostrar los resultados de nuestra calculadora.

¡Estamos listos para comenzar a activar esta calculadora! Vamos a ir al controlador y le agregaremos estos valores:

```
1 $scope.valor1 = 0;
2 $scope.valor2 = 0;
```

Con ellos le asignamos un valor inicial a las casillas de texto, si abres tu documento veras que ahora tienen un valor de “0”

Agregaremos ahora un método dentro de nuestro controlador. Como hemos mencionado antes, un método no es más que una función dentro de un objeto, en este caso , llamaremos al método “sumar” y como queremos que tenga acceso a los datos del documento lo crearemos dentro de \$scope

```
1 $scope.sumar= function (){
2     $scope.resultado = $scope.valor1 + $scope.valor2;
3 }
```

Dentro de un método puedes insertar cualquier instrucción que desees ejecutar cuando se invoque, en este caso estoy sumando los dos valores de las casillas de texto y asigno el valor resultante dentro de la variable “resultado” para que la muestre la expresión del HTML. En todos los casos use \$scope para tener acceso a los valores del documento.

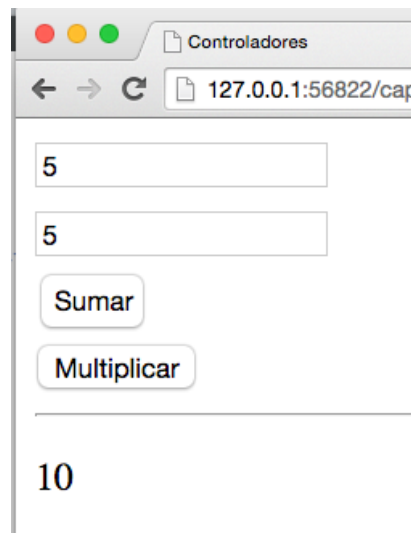
A diferencia de las variables y propiedades que se asignan al momento de cargar el documento, los métodos no muestran sus resultados hasta que son invocados, por eso, aunque mi método suma los valores de las casillas, aunque cambie sus valores en el documento, no se mostrara el resultado del cálculo.

Para invocar nuestro método vamos a usar una directiva especial de AngularJS que activa conductas al hacer click sobre un elemento. Modifica el botón sumar con este nuevo código:

```
1 <button ng-click="sumar()"> Sumar </button>
```

En este caso, la directiva “ng-click” se activa cada vez que le hagas click al botón sumar y como notarás , la directiva esta activando el método “sumar” que acabamos de crear. ¡Pon especial atención a la sintaxis! : los métodos siempre, SIEMPRE, se deben invocar seguidos de paréntesis, de lo contrario no se activaran correctamente.

Tu aplicación ya esta funcional, al incluir cualquier número en las casillas obtendrás un resultado al presionar el botón sumar



Un controlador puede tener una cantidad indefinida de métodos, si quisieras incluir por ejemplo un nuevo método para multiplicar las mismas cifras, solo agrega el código de este nuevo método en tu controlador:

```
1 $scope.multiplicar = function (){  
2   $scope.resultado = $scope.valor1 * $scope.valor2;  
3 }
```

Recuerda que para ejecutar este método debes también relacionarlo en el html, en este caso, modificaremos el botón “multiplicar” para que ejecute el nuevo método cada vez que se le haga click

```
1 <button ng-click="multiplicar()"> Multiplicar </button>
```

Ya tienes tu primera calculadora básica en AngularJS ¡Y solo te tomó unas cuantas líneas de código!

El código final de tu controlador con métodos quedaría así:


```
1  controladorApp.controller("controladorConMetodos", function($scope){
2
3      $scope.valor1 = 0;
4      $scope.valor2 = 0;
5
6      $scope.sumar= function (){
7          $scope.resultado = $scope.valor1 + $scope.valor2;
8      }
9
10     $scope.multiplicar = function (){
11         $scope.resultado = $scope.valor1 * $scope.valor2;
12     }
13
14     })
```

Directivas

Las directivas de AngularJS son elementos que se insertan en el código HTML para habilitar conductas complejas.

Al utilizar una directiva dentro de una etiqueta HTML le indicas al compilador de AngularJS que debe agregar relaciones, eventos y valores con la aplicación que esta ejecutando.

Las directivas te permiten habilitar conductas sofisticadas con solo agregar unas cuantas propiedades extra a una etiqueta, por esta razón muchos desarrolladores dicen que las directivas de AngularJS nos permiten enseñarle nuevos trucos al HTML.

Agregar Directivas

Desde el momento que relacionas el archivo JavaScript de AngularJS con tu documento tienes acceso a las directivas básicas del Framework.

La forma más común para agregar una directiva es incluyéndola como un atributo más dentro de cualquier etiqueta html de un documento, a manera de ejemplo general, la estructura de una directiva angular es:

```
1 <etiqueta DIRECTIVA_ANGULAR="VALOR">
2 </etiqueta>
```

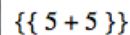
Por regla general las directivas de AngularJS usan el prefijo “ng-” antes de su nombre lo que las hace fáciles de identificar, aplicando este concepto al ejemplo anterior, es más posible que encuentres algo como esto:

```
1 <etiqueta ng-DIRECTIVA="VALOR">
2 </etiqueta>
```

Las directivas son absolutamente fundamentales y es prácticamente imposible que encuentres una aplicación AngularJS que no las utilice, de hecho, para inicializar una aplicación angular necesitas hacerlo a través de la directiva “ng-app”. Vamos a utilizar como base el siguiente código para crear una aplicación AngularJS básica.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>Directivas</title>
6      </head>
7      <body>
8
9          <p>{{ 5 + 5 }}</p>
10
11          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular\
12 ar.min.js"></script>
13
14      </body>
15 </html>
```

En este código tenemos enlazado el documento con AngularJS y estamos desplegando una expresión básica, técnicamente debería funcionar, pero si lo pruebas en tu navegador obtendrás como resultado el código sin procesar de la expresión.



Browser console output showing the unprocessed AngularJS expression: `{{ 5 + 5 }}`

Esto se debe, a que aunque AngularJS esta disponible en el documento no se encuentra la directiva que define el área en que trabajarás la aplicación; para agregar esta directiva solo tienes que incluir un nuevo atributo en cualquier etiqueta del documento, por ejemplo en la etiqueta `<body>` :

```
1  <body ng-app>
```

Ahora todos los elementos que se encuentren dentro de `<body>` serán procesados como parte de la aplicación AngularJS, como nuestra expresión se encuentra allí dentro, al abrir tu documento ahora encontrarás que se está ejecutando correctamente.



Browser console output showing the result of the AngularJS expression: `10`

Directivas con valores

En el ejemplo anterior, aprendimos que con solo declarar la directiva “ng-app” en una etiqueta, podíamos activar AngularJS, sin embargo, aunque sea técnicamente posible, esto es una mala práctica, si queremos hacer una aplicación que vaya más allá de los procesos más rudimentarios, es indispensable asignarle un nombre. Para asignarle un nombre a nuestra aplicación conservaremos la directiva en el mismo lugar, pero esta vez le incluiremos un valor, reemplazaremos la directiva actual por esta:

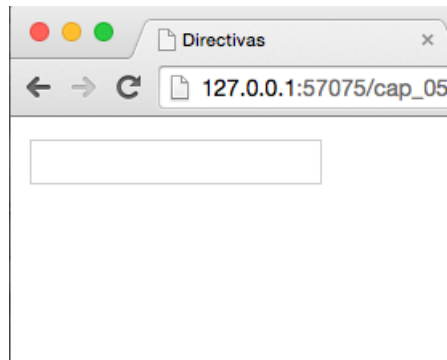
```
1 <body ng-app="directivas">
```

En este caso acabamos de asignarle un valor a la directiva “ng-app” para que nuestra aplicación sea identificada como “directivas”. La mayoría de las directivas reciben valores y estos le permiten adjuntar, procesar o relacionar diferentes elementos o datos de tu aplicación, dependiendo del valor que se le agregue a una directiva es posible que obtengamos diferentes comportamientos. Cada directiva requiere valores distintos según su configuración individual.

Retomemos el código de nuestro ejemplo, vamos a agregarle algunos elementos extra para probar el comportamiento de algunas directivas y sus valores, el código de nuestro ejemplo debería ser así:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Directivas</title>
6   </head>
7   <body ng-app="directivas">
8
9     <input type="text">
10
11     <p>{{ textoModelo }}</p>
12
13     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular\
14 ar.min.js">
15     </script>
16
17     <script>
18       var directivaApp = angular.module('directivas', []);
19
20     </script>
21   </body>
22 </html>
```

Los cambios son mínimos, pero importantes: primero he agregado una casilla de texto vacía y cambie el valor que usa la expresión. También, ahora que la aplicación tiene un nombre, incluí una declaración de la misma para que AngularJS pueda relacionar todos los elementos y todo se ejecute correctamente. Al probar este código deberías tener únicamente una casilla de texto vacía sin funcionalidad.

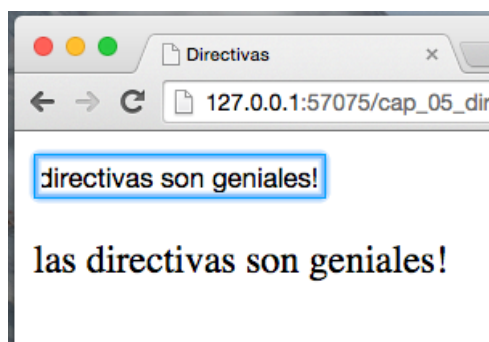


Usaremos ahora una directiva llamada “ng-model” que define modelo o dato dentro de la aplicación. En este caso lo usaremos para relacionar los datos que se encuentran en la casilla de texto, con el valor “textoModelo” que nuestra aplicación esta representando en una expresión.

Localiza la casilla de texto y le agrega la nueva directiva, también le asignaremos un valor “textoModelo”

```
1 <input type="text" ng-model="textoModelo">
```

Con esta simple acción, la directiva “ng-model” esta creando un objeto llamado “textoModelo” dentro de tu aplicación, también esta relacionando cualquier dato que se encuentre en la casilla de texto y además este proceso lo hace en tiempo real, o sea que cualquier cambio que tenga la casilla de texto, se reflejara en “textoModelo”. Este efecto se puede notar si escribes algo en la casilla de texto, cualquier dato que incluyas allí, gracias a la acción de “ng-model” se reflejara la información que muestra tu expresión.



Directivas AngularJS y estándares W3C

Las directivas tienen una implementación simplemente elegante, te permiten mejorar tu documento sin necesidad de reinventar la rueda, AngularJS utiliza la arquitectura original de HTML aprovechando los atributos del HTML para crear sus aplicaciones.


Los atributos son parte integral de HTML desde sus inicios y AngularJS los utiliza como marcadores para agregar conductas o valores en el documento al momento de cargarlo. Esta técnica permite que las aplicaciones AngularJS sean compatibles prácticamente con cualquier navegador web moderno.

A pesar de todo, si eres de los que se obsesionan con los standards (¡Y te felicito por eso!) es posible que encuentres un problema con AngularJS: Las directivas que hemos usado no pasan la validación W3C.

Tomemos un ejemplo básico de una aplicación AngularJS

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>Directivas</title>
6      </head>
7      <body ng-app>
8
9          <p>{{ 5 + 5 }}</p>
10
11          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular\
12 ar.min.js"></script>
13
14      </body>
15  </html>
```

Aunque el código está bien formado y funciona perfectamente en un navegador, si lo validamos en el sitio oficial de la W3C (http://validator.w3.org/#validate_by_input) encontraremos que no es un documento standard


Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents

Jump To:
[Notes and Potential Issues](#)
[Validation Output](#)

Error found while checking this document as HTML5!


Result:	1 Error, 2 warning(s)
Source :	<pre> <!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>Directivas</title> </head> <body ng-app> <p>{{ 5 + 5 }}</p> <script src="https://ajax.googleapis.com/ajax/libs/angularjs </pre>

Esto se debe a que las reglas de HTML5 dicen que para mantener el orden dentro del lenguaje no puedes agregar atributos arbitrarios dentro de una etiqueta. En otras palabras, las directivas, que a los ojos del validador son solo un atributo más, no son parte de la definición de HTML.

La solución es increíblemente sencilla, HTML5 incorpora una opción para incluir nuevos atributos, solamente debe agregarse el prefijo “data-”. En nuestro caso deberíamos reemplazar cualquier directiva que comience con “ng-” por “data-ng-” para que nuestro documento sea valido. Reemplaza la etiqueta <body> del ejemplo original por esta:

1 <body data-ng-app>

Ahora tu código cumple al 100% con los standards de la W3C.. ¡Como debe ser!


Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents

[Jump To:](#)
[Notes and Potential Issues](#)
[Congratulations · Icons](#)

This document was successfully checked as HTML5!

Result:	Passed, 2 warning(s)
Source :	<pre> <!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>Directivas</title> </head> <body data-ng-app> <p>{{ 5 + 5 }}</p> <script> </pre>

Sobre la funcionalidad de tu aplicación no debes preocuparte AngularJS toma en cuenta esta regla y tiene total compatibilidad con las directivas que usan el prefijo “ng-” de la misma forma como las que usan la versión standard “data-ng-”

Directivas Comunes

A lo largo de este libro examinaremos en detalle muchas directivas en conjunto con diferentes elementos y temas, sin embargo te mostraré algunas de las más comunes para que te familiarices con ellas y comprendas más fácilmente su rol dentro de los ejemplos que veremos más adelante.

ng-app

Esta directiva inicializa tu aplicación, se incluye generalmente en las etiquetas `<body>` o `<html>` para asegurarse que todo el documento esta cubierto por la aplicación.

Se recomienda siempre asignarle como valor el nombre de la aplicación para garantizar compatibilidad y soporte para módulos.

Puedes utilizar esta directiva al inicio de tu aplicación tal como se muestra acá:

```
1 <body ng-app="miAplicacion">
```


ng-controller

Una vez creada una aplicación es muy probable que comiences a agregar controladores. Esta directiva te permite relacionar un controlador de la aplicación con su área de acción.

Puedes incluir esta directiva en cualquier etiqueta, pero generalmente se incluye en elementos contenedores como por ejemplo los `<div>`. El controlador asignado solo tendrá efecto dentro de la etiqueta en que se incluya. El valor que se le agrega a “ng-controller” es el nombre del controlador que se desea relacionar con la etiqueta.

Un uso común para esta directiva puede ser similar a este:

```
1 <div ng-controller="miControlador">
2 </div>
```

ng-model

En un documento no solo necesitas mostrar texto, también tendrás entradas de información de los usuarios, el mejor ejemplo de esto son los formularios. La directiva “ng-model” tiene la capacidad de vincular los valores de un elemento con un dato dentro de tu aplicación. Su uso más común es justamente en los formularios recopilando los datos que inserta el usuario e incorporándolos en la aplicación. El valor que se le asigna a esta directiva es el nombre que tendrá un valor dentro de la aplicación.

Para aplicar “ng-model” puedes usar una estructura similar a esta:

```
1 <input type="text" ng-model="miValorDelUsuario" >
```

ng-bind

Esta directiva no es tan común como las demás, pero vale la pena mencionarse porque la puedes necesitar en algunos casos especiales. La directiva “ng-bind” básicamente vincula un valor de la aplicación y lo representa dentro del documento, cumple así la misma función que una expresión básica. Por ejemplo, asumiendo que tienes una variable llamada “dia” con el valor “lunes” ambos métodos te darían el mismo resultado

```
1 <p>{{ dia }}</p>
2
3 <p ng-bind="dia"> </p>
```

La única diferencia es que en el caso de que tu aplicación tenga problemas de carga o el equipo en el que funciona es muy lento, en caso de que uses la primera expresión en la pantalla de inicio, corres el riesgo que se muestre por unos segundos el código original



Por el contrario, gracias a que la segunda expresión incluye sus valores dentro de un atributo de la etiqueta `<p>` hasta que AngularJS no compile el documento, el usuario solo vera un elemento vacío sin ningún código extraño.

En el mundo real, te recomiendo que siempre que puedas uses la sintaxis normal de doble llave para representar las expresiones, eso te facilitara leer y comprender mejor que ocurre en tu código, así como asegurar la compatibilidad, sin embargo, si un día tienes una pagina de inicio que se niega a desplegar contenidos desde el primer milisegundo ¡ya sabes que truco usar!

Directivas personalizadas

Al igual que la mayoría de los elementos de AngularJS, puedes personalizar directivas para que se ajusten a las necesidades específicas de tu aplicación.

El objetivo de personalizar una función es operacionalizar conductas complejas o repetitivas para que se conviertan en simples de elementos HTML.

Exploraremos el uso de las directivas personalizadas en un nuevo ejemplo y comenzaremos con un documento en blanco:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>Directivas</title>
6      </head>
7      <body ng-app="directivas">
8
9          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.min.js">
10
11          </script>
12
13          <script>
14              var directivaApp = angular.module('directivas', []);
```

```
15
16     </script>
17 </body>
18 </html>
```

En este documento, tenemos una aplicación AngularJS instalada y lista para trabajar, pon atención como ya he declarado la aplicación y estoy usando la variable “directivaApp” para acceder a ella.

Creando una directiva personalizada

Una vez declarada la aplicación usaremos la variable que usamos como su instancia , en este caso la variable “directivaApp” para crear una nueva directiva. Coloca este código luego de la declaración de la aplicación:

```
1  directivaApp.directive('usuario', function() {
2      var valoresInternos = {};
3      valoresInternos.template = “¡Bienvenido!”;
4
5      return valoresInternos;
6
7  });
```

Aquí estamos creando una directiva dentro de la aplicación actual a través del método “directive”. Estamos enviando dos argumentos a la nueva directiva: “usuario” que será el nombre de la directiva y una función que será el código que se ejecute cuando la invoquemos.

Dentro de esta directiva incluí algunos valores básicos, revisémoslos línea por línea:

```
1  var valoresInternos = {};
```

Esta variable llamada “valoresInternos” almacenara un objeto, puedes ponerle el nombre que quieras pero es importante crear un objeto en este punto porque las directivas siempre devuelven un objeto con algunas propiedades especificas, por ejemplo la de la siguiente línea:

```
1  valoresInternos.template = “¡Bienvenido!”;
```

Aquí estamos agregando una propiedad especializada de las directivas, “template” es el texto que se mostrará dentro de la etiqueta en que se use esta directiva, por ahora este texto es lo único que va a desplegar nuestra directiva, por eso cerramos la función entregando el resultado final con el objeto requerido para ser procesado:

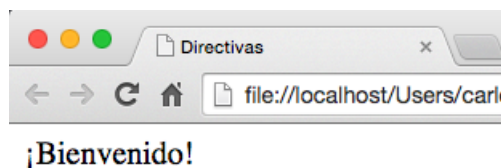
```
1 return valoresInternos;
```

Aplicando directivas personalizadas en un documento

Tenemos ya una directiva personalizada lista para usar y AngularJS nos ofrece muchas opciones para agregarla al documento, la forma más común sería incluirla como un argumento dentro de una etiqueta HTML, por ejemplo, este código activa nuestra nueva directiva:

```
1 <p usuario > </p>
```

Desde la perspectiva de AngularJS, al incluir el atributo “usuario” estas adjuntando una directiva dentro de la etiqueta <p> y las instrucciones que le dimos fue que a través de la propiedad “template” escriba el valor “¡Bienvenido!”.



Nada mal para empezar, pero, ¿Sabías que las directivas de AngularJS no están limitadas a los atributos? Puedes, por ejemplo obtener el mismo resultado usando una nueva etiqueta HTML con el nombre de la directiva, de esta forma:

```
1 <usuario></usuario>
```

Ambas formas son equivalentes para AngularJS y de hecho son muy populares entre los desarrolladores por su flexibilidad, existen además otras formas menos populares pero igualmente validadas como utilizar clases de CSS o comentarios HTML.

Si quieres que tu controlador utilice exclusivamente una forma para agregarse al código HTML, puedes utilizar la propiedad “restrict”. Por ejemplo si quisiera que mi directiva “usuario” solamente se pueda invocar cuando llamo la etiqueta personalizada <usuario> debería crearla así

```
1  directivaApp.directive('usuario', function() {
2      var valoresInternos = {};
3
4      valoresInternos.restrict = 'E';
5      valoresInternos.template = "¡Bienvenido!";
6
7      return valoresInternos;
8
9  });
```

En este caso he dejado el código igual salvo la diferencia que incluí la propiedad “restrict” con el valor “E” lo que para AngularJS significa que esta directiva esta restringida para activarse únicamente como elemento. También están las opciones como “A” o “C” para restringir la activación solamente a atributos o clases respectivamente.

Agregando valores a las directivas personalizadas

Las directivas personalizadas pueden tener toda la complejidad que requieras: conectarse a bases de datos externas, usar valores internos de la aplicación, interactuar con otras directivas y muchas otras capacidades de AngularJS que aprenderemos en este libro, por ahora vamos a comenzar enviando valores adicionales a la directiva.

Continuando con la directiva “usuario” que creamos hace un momento, podemos por ejemplo darle soporte a valores dinámicos, por ejemplo, supongamos que estamos usando la directiva en forma de etiqueta y queremos enviar allí el nombre del usuario para que le de un saludo personalizado, comenzamos agregándole un atributo adicional para almacenar el nombre del usuario

```
1  <usuario nombre="Erik"> </usuario>
```

Ahora, para capturar estos valores dentro de la directiva agregamos una nueva propiedad llamada “scope” que es básicamente un objeto que nos permite capturar los valores de los parámetros . En el caso específico de nuestro controlador el código sería así:

```
1  valoresInternos.scope = {
2      nombreUsuario : '@nombre',
3  }
```

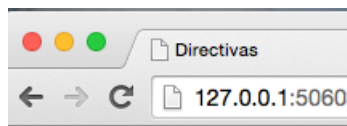
Estamos tomando el objeto “valoresInternos” que funciona como un contenedor de las propiedades de la directiva y le insertamos la propiedad “scope”, dentro de ella tenemos un objeto con una propiedad llamada “nombreUsuario” que almacenara un valor y estará disponible dentro de la directiva. El valor que se le asigna es en este caso “@nombre” que

es la nomenclatura que usaremos para capturar valores en la directiva: el signo @ seguido del nombre del método que deseas leer.

Solo queda agregar el valor que acabamos de capturar dentro del texto que muestra nuestra directiva, específicamente, dentro de la propiedad "template" con la sintaxis normal de una expresión. En nuestro ejemplo el código sería así

```
1 valoresInternos.template = "¡Bienvenido {{nombreUsuario}}!"
```

Tenemos todo listo para mostrar un resultado y al ejecutar este código, la aplicación ya puede integrar el nuevo valor dinámico



Bienvenido Erik!!

El código completo de nuestro ejemplo quedaría así:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Directivas</title>
6   </head>
7   <body ng-app="directivas">
8
9     <usuario nombre="Erik" apellido="Porroa"> </usuario>
10
11
12
13     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular\
14 ar.min.js">
15   </script>
16
17   <script>
18     var directivaApp = angular.module('directivas', []);
19
20     directivaApp.directive('usuario', function() {
```

```
21
22         var valoresInternos = {};
23
24
25         valoresInternos.scope = {
26             nombreUsuario : '@nombre',
27             apellidoUsuario: '@apellido'
28         }
29
30         valoresInternos.restrict = 'E';
31         valoresInternos.template = "¡Bienvenido {{nombreUsuario}} {{\
32 apellidoUsuario}}!";
33
34         return valoresInternos;
35
36     });
37
38
39     </script>
40 </body>
41 </html>
```

Aproveché para incluirle un nuevo atributo de “apellido” usando los mismos pasos aplicados anteriormente, ahora nuestra directiva personalizada recibe dos valores dinámicos y los despliega sin problema en el documento.



Modificar estilos con AngularJS

No importa si tu aplicación es simple o compleja, si es una tienda o un experimento, inevitablemente debes utilizar componentes gráficos para representar los datos y procesos.

En AngularJS creas aplicaciones web y al hablar de componentes gráficos en este entorno, estamos hablando de CSS que es el lenguaje dedicado a administrar todo el aspecto visual del HTML.

A pesar de ser un framework enfocado específicamente a la parte lógica de una aplicación, AngularJS puede manipular el CSS y propiedades visuales de una aplicación para representar estados o resultados de procesos.

En este capítulo vamos a crear un ejemplo donde exploraremos las principales directivas y las formas de controlar con ellas las propiedades visuales de tus aplicaciones.

Utilizaremos este código como base para comenzar a trabajar

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="utf-8">
6      <title>Estilos</title>
7      <script src="angular.min.js"></script>
8      <style>
9          img {
10             display: block;
11             margin: auto
12         }
13         .preguntas {
14             width: 400px;
15             margin: 30px auto;
16             background: silver;
17             padding: 20px;
18             text-align: center
19         }
20     </style>
21 </head>
22
23 <body ng-app="estilosApp">
```



```

24     <div ng-controller="controladorEstilos">
25
26         
28
29         <div id="contenedorCaja">
30             
32         </div>
33         <div class="preguntas">
34             <button ng-click="opcionA()">Opción A</button>
35             <button ng-click="opcionB()">Opción B</button>
36             <hr>
37
38             <button ng-click="cambiarEstilo()">Cambiar Estilo</button>
39             <button ng-click="cambiarClase()">Cambiar Clase</button>
40
41             <hr>
42             <button ng-click="ocultarCaja()">Ocultar Caja</button>
43             <button ng-click="mostrarCaja()">Mostrar Caja</button>
44         </div>
45
46     </div>
47
48
49     <script>
50         var misEstilos = angular.module('estilosApp', []);
51
52         misEstilos.controller('controladorEstilos', function ($scope) {
53
54             $scope.opcionA = function () {
55
56             }
57
58             $scope.opcionB = function () {
59
60             }
61
62             $scope.cambiarEstilo = function () {
63
64             }
65

```

```

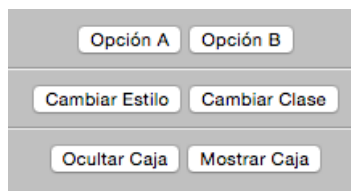
66         $scope.cambiarClase = function () {
67
68         }
69
70         $scope.ocultarCaja = function () {
71
72         }
73         $scope.mostrarCaja = function () {
74
75         }
76
77     })
78     </script>
79 </body>
80
81 </html>

```

En este código inicial ya tenemos una aplicación básica y algunos elementos de programación predefinidos. El documento también incluye elementos básicos de CSS, que permiten desplegar correctamente una serie de imágenes y botones que usaremos para los ejercicios de este capítulo.

Dentro de esta aplicación encontrarás un controlador llamado “controladorEstilos” que esta administrando todos los elementos visuales de la aplicación, dentro de él he creado serie de métodos, que por ahora no ejecutan ninguna acción.

Cada uno de estos métodos está relacionado a un botón y ya están debidamente vinculados para que al hacer click sobre ellos se active su respectivo controlador.



Te invito a que antes de continuar esta lectura, analices el código y explores los elementos que lo componen para que comprendas sin problema los ejercicios que están a continuación.

Manipular propiedades CSS

Esta es una de las capacidades de AngularJS que más vas a utilizar, piensa en este escenario: Cuando un usuario selecciona una opción incorrecta en tu aplicación ¿Cómo le muestras que lo hizo bien o mal? Una opción sencilla y fácil de comprender es simplemente cambiar de color ese elemento para que el usuario encuentre donde esta el problema.

Un caso simple, pero muy común, al igual que este, muchas veces te verás en la necesidad de modificar de alguna forma las propiedades gráficas de los elementos en pantalla, para este tipo de situaciones tienes la directiva llamada “ng-style”. Esta directiva inyecta valores CSS directamente en el elemento en que se encuentre y te permite modificar cualquier propiedad de CSS que desees.

En este ejercicio vamos a modificar la apariencia del menú de la aplicación cambiando sus propiedades de CSS.

Antes de comenzar, vamos a modificar el código, primero localiza el <div> con la clase “preguntas” y agrega allí la directiva “ng-style”

```
1 <div class="preguntas" ng-style="estilosAngular">
```

En este caso he incluido la directiva con el valor “estilosAngular”, este valor es el que nos permitirá inyectar propiedades en el menú. Busca el botón “cambiar Estilo” si revisas el código notarás ya tiene incluida una directiva

```
1 <button ng-click="cambiarEstilo()">Cambiar Estilo</button>
```

Al hacer click sobre este botón activaremos un método llamado “cambiarEstilo” y vamos a utilizar este método para que al activarse modifique los valores de la variable “estilosAngular” y así, cambie la apariencia del menú.

Localiza el método “cambiarEstilo”, que por ahora esta en blanco

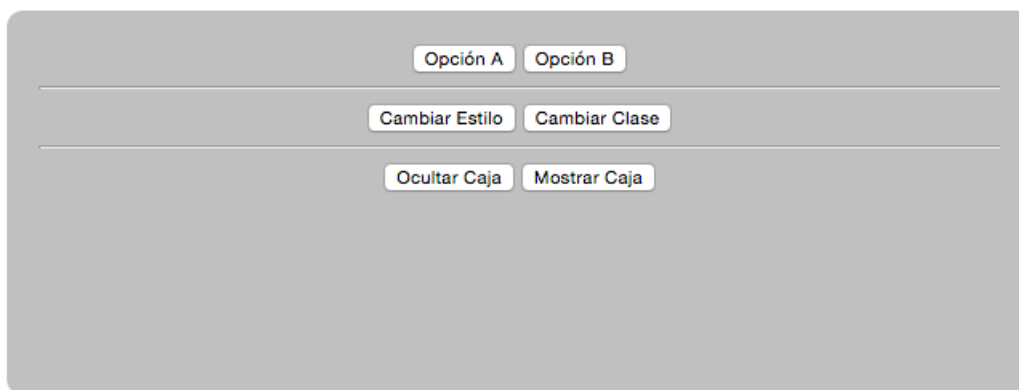
```
1 $scope.cambiarEstilo = function () {
2 }
```

El código que pongas dentro de este método, se ejecutará al presionar el botón de “cambiar estilo” así que allí incluiremos los nuevos valores CSS, inserta este código dentro del método

```
1 $scope.estilosAngular = {
2     width: '600px',
3     height: '200px',
4     borderRadius: '10px'
5 }
```

Como puedes notar, esto define un grupo de propiedades CSS para la variable “estilosAngular”, en AngularJS se utiliza el formato JSON para inyectar las propiedades de CSS en un elemento, en otras palabras, necesitas crear un objeto que contenga las propiedades de CSS con sus respectivos valores, utilizando este patrón, puedes incluir cualquier cantidad de propiedades gráficas.

Ahora en tu aplicación cada vez que presiones el botón “cambiar estilos” defines las propiedades gráficas en una variable y la directiva “ng-style” se encarga de inyectarlas dentro del html, el resultado será similar a este:



Si te fijas bien, la propiedad “borderRadius” como la usamos aquí es ligeramente diferente a “border-radius”, la forma en que se usa tradicionalmente en CSS. Esto es porque AngularJS usa la nomenclatura de JavaScript para estas propiedades, puedes encontrar una lista completa de todas ellas en la documentación oficial del W3C

<http://www.w3.org/TR/DOM-Level-2-Style/css.html#CSS-extended>

Asignar clases

Ya aprendimos a modificar las propiedades de manera individual, pero AngularJS no se limita a esto, también puedes asignar grupos de propiedades a través de las Clases de CSS. Manipular propiedades puede ser útil en caso de que los cambios sean menores y no se repitan, pero para hacer cambios más complejos y escalables debemos usar las clases.

AngularJS tiene la directiva “ng-class” que se encarga de asignar clases de manera dinámica. Vamos a incluir esta directiva en el el menú de nuestra aplicación, el resultado será similar a este:

```
1 <div class="preguntas" ng-style="estilosAngular" ng-class="clasesAngular" >
```

Aprovecho este ejemplo para que notes que puedes incluir diferentes directivas relacionadas con las propiedades gráficas sin que esto afecte su funcionamiento.

En este ejemplo utilizamos el valor “clasesAngular”, la conducta de la directiva “ng-class” será asignarle una clase de acuerdo al valor que almacene esa variable.

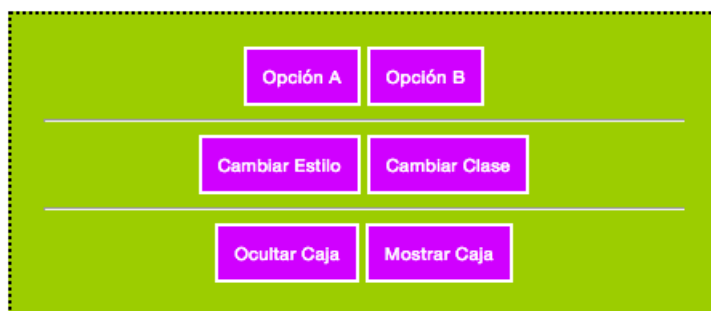
Vamos a utilizar el botón “Cambiar clase” para modificar las propiedades del contenedor, este elemento se encuentra relacionado con el método “cambiarClase()” así que busca el método “\$scope.cambiarClase” en el código de la aplicación.

En el código base del ejercicio incluí una clase para este contenedor llamada “segundoEstilo”, para asignarle esta clase al contenedor agrega esta línea de código al método “\$scope.cambiarClase”

```
1 $scope.clasesAngular = 'segundoEstilo'
```

Al igualar esta variable con el nombre de la clase “segundoEstilo” vas a asignarla directamente al elemento HTML y por ende modificar su apariencia de inmediato.

El resultado será similar a este una vez que presiones el botón



Ocultar y mostrar elementos

Al crear tus aplicaciones, en muchas ocasiones necesitarás únicamente ocultar o mostrar componentes sin modificar ninguna de sus propiedades. Por ejemplo para mostrar ventanas con mensajes de error u ocultar secciones de acuerdo a las preferencias del usuario.

Este tipo de acciones puedes realizarlas usando las directivas que acabamos de estudiar a través de la modificación de estilos o asignación de clases, pero ya el proceso de ocultar y mostrar elementos es particularmente común en una aplicación, AngularJS tiene dos directivas que controlan directamente las opciones de visualización para hacer el proceso más ágil y rápido.

Vamos a disponer de dos directivas “ng-hide” y “ng-show”, su rol es ocultar o mostrar un elemento respectivamente.

Estas directivas trabajan a partir de valores lógicos o booleanos, en otras palabras se activan o desactivan de acuerdo a sus valores. A manera de esquema si quieres que una de estas directivas se active le asignas un valor que sea verdadero

```
1 DIRECTIVA = true
```

Por el contrario, el valor opuesto desactiva la acción de la directiva

```
1 DIRECTIVA = false
```

Aplicando este principio a un ejemplo concreto, supongamos que tienes en un elemento la directiva “ng-hide” la función de esta directiva es OCULTAR un elemento, entonces si le asignas un valor VERDADERO oculta el elemento. Por otro lado, si le asignas un valor FALSO la directiva se desactiva y por ende no oculta el elemento.

Puede que al inicio te parezca algo confuso este funcionamiento, pero la idea detrás de estas directivas es ofrecerte la mayor cantidad de opciones para adaptarse a la lógica y necesidades de tu aplicación. En esta tabla se detallan todas las posibles combinaciones y resultados que puedes obtener

DIRECTIVA	VALOR	RESULTADO
ng-show	TRUE	Muestra elemento
ng-show	FALSE	Oculto elemento
ng-hide	TRUE	Oculto elemento
ng-hide	FALSE	Muestra elemento

Veamos esta mecánica aplicada en nuestra aplicación, vamos a aplicar la directiva “ng-hide” al <div> con el identificador “contenedorCaja”

```
1 <div id="contenedorCaja" ng-hide="ocultarMenu">
2   
4 </div>
```

Dentro de la directiva asignaremos el valor “ocultarMenu”, según la lógica de AngularJS, este valor será relacionado a una variable en la aplicación con ese nombre.

En este punto de la aplicación, esa variable no esta definida y por ende no tiene ningún valor asignado, así que será interpretada como un valor “false”, esto desactiva la directiva “ng-hide” y como resultado no oculta el <div> en que se encuentra.

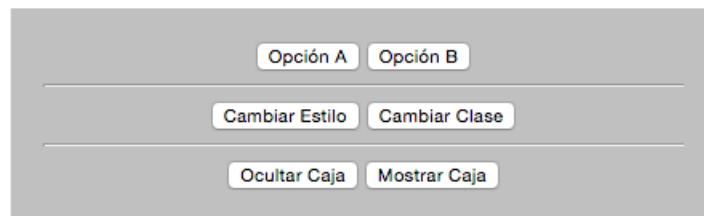
Ahora, vamos a modificar el método “ocultarCaja” y dentro de él, vamos a asignar un valor “true” para la variable “ocultarMenu”

```

1  $scope.ocultarCaja = function () {
2      $scope.ocultarMenu = true
3  }

```

Debido a que este método se activa al hacer click en el botón “ocultar caja”, al presionar dicho botón, la variable “ocultarMenu” se vuelve verdadera, lo que significa que ahora sí vamos a activar la directiva “ng-hide” y como la función de esta directiva es ocultar el elemento en el que se encuentra, veras como la imagen de la caja desaparece.



Para revertir este efecto, debemos volver a deshabitar la directiva. Busca el método “mostrar-Caja”, allí vamos a asignarle el valor “false” a la variable “ocultarMenu”

```

1  $scope.mostrarCaja = function () {
2      $scope.ocultarMenu = false
3  }

```

Este método se activa al presionar el botón “mostrar Caja” y al hacer click allí, cambiamos una vez más el valor que controla a “ng-hide”, al definirlo como falso, desactivamos la directiva y así deja de ocultar el componente.

Te recomiendo que al terminar este ejercicio, para que comprendas mejor estas directivas, pruebes reemplazar “ng-hide” por “ng-show” y modifiques los valores según la tabla que te mostré al inicio de esta sección para te familiarices con el funcionamiento de estas útiles directivas.

Modificar imágenes

Hasta ahora hemos visto propiedades CSS, pero uno de los elementos más poderosos de una interfaz es la imagen y en esta sección aprenderás a modificarlas usando AngularJS.

Para cambiar una imagen por otra utilizaremos la directiva “ng-src” que reemplaza la propiedad “src”, esto nos permite usar variables para definir la ruta de la imagen que se muestra en el documento.

Vamos a aplicar este concepto en el ejemplo que estamos trabajando, para esto, debes primero modificar el código HTML

```
1 
```

En este caso usaremos la directiva “ng-src” y usaremos como valor una expresión llamada “imagenActiva”, el valor que se encuentre en esta expresión, será la ruta de la imagen.

Como puedes notar en esta etiqueta he eliminado del todo el atributo “src”, a partir de ahora la directiva de AngularJS se encarga de administrar la imagen dinámicamente.

Por ahora, este elemento no mostrará ninguna imagen porque aun no hay ningún valor asignado para la expresión, así que busca el controlador de esta aplicación y agrega esta línea de código dentro de él

```
1 $scope.imagenActiva = 'http://www.manualdelguerrero.com/angularjs/estilos/img/in\
2 certidumbre.png'
```

Ahora cuando el controlador se inicie lo primero que hará será asignar el valor de la ruta a una imagen almacenada en internet.

El siguiente paso es modificar esta imagen de acuerdo a la elección del usuario, usaremos para esto los botones “Opción A” y “Opción B”, los cuales, al igual que el resto de los botones de esta aplicación ya están relacionados a sus respectivos métodos.

Lo que haremos en este ejercicio es asignar un valor diferente para la variable “imagenActiva” según el botón presionado, comenzaremos modificando el método “\$scope.opcionA”

```
1 $scope.opcionA = function () {
2     $scope.imagenActiva = 'http://www.manualdelguerrero.com/angularjs/estilos/i\
3 mg/cat1.png'
4 }
```

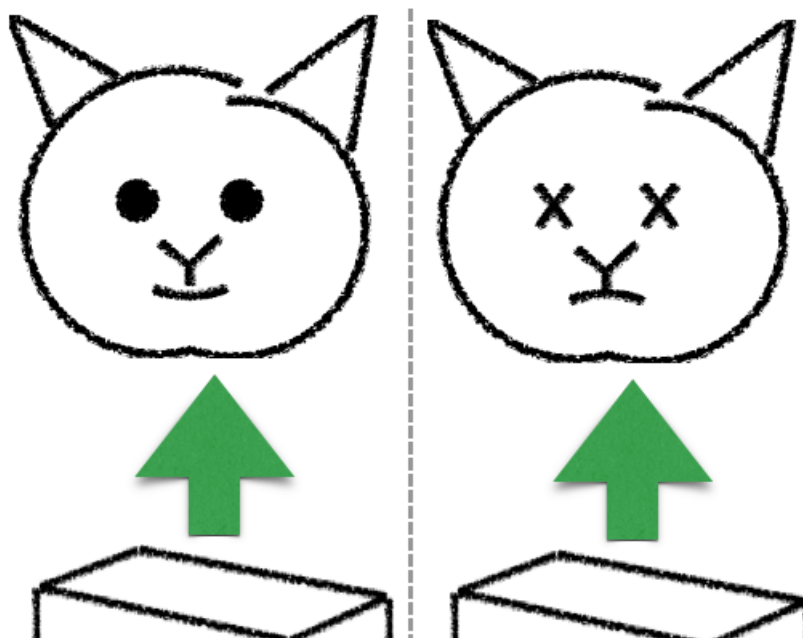
Una vez hecho este cambio, replicaremos la misma mecánica en el método “\$scope.opcionB”


```

1 $scope.opcionB = function () {
2     $scope.imagenActiva = 'http://www.manualdelguerrero.com/angularjs/estilos/i\
3 mg/cat2.png'
4 }

```

Ahora al presionar cada botón verás como se modifica la imagen de inmediato



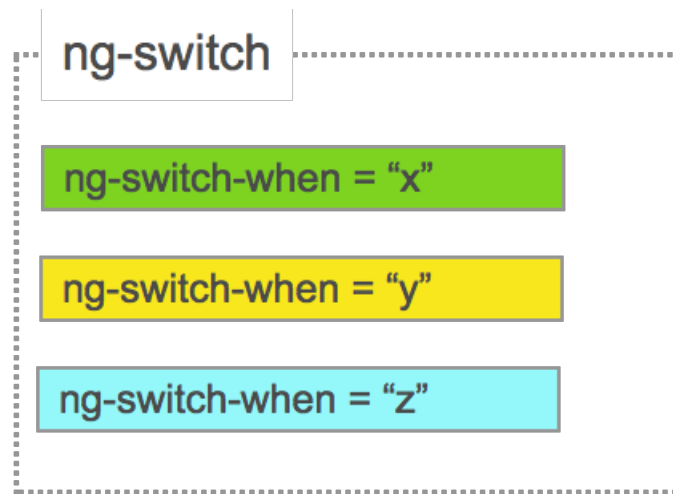
Mostrar elementos condicionalmente

Para finalizar este capítulo, quiero mostrarte una directiva con un comportamiento muy interesante. En los ejercicios anteriores básicamente asignamos valores para modificar la apariencia, pero la directiva “ng-switch” trabaja diferente: evalúa una condición y dependiendo de su valor puede mostrar u ocultar diferentes elementos de HTML

Esta directiva utiliza dos partes para mostrar el contenido de manera condicional y para ello, primero necesitamos un contenedor que englobe todas las opciones, en este caso dicho contenedor será la directiva “ng-switch” a la cual le asignaremos un valor que será evaluado más adelante.

Luego, dentro del elementos que contiene esta directiva se incluyen las diferentes opciones para mostrar, cada una de ellas usando la directiva “ng-switch-when” la cual evalúa el valor de la variable definida en “ng-switch” y solamente muestra el bloque que coincida.

A manera de ejemplo general, en este gráfico se muestra un esquema del funcionamiento de la directiva “ng-switch”



En este ejercicio vamos a utilizar dicha directiva para mostrar un mensaje distinto de acuerdo a la opción que elige el usuario, comenzaremos incluyendo este código en el HTML justo después de <div> con el id “contenedorCaja”

```
1 <div ng-switch="opcion">
2   <h2 ng-switch-when="a"> ¡Gato Feliz! </h2>
3   <h2 ng-switch-when="b"> Mal día para ser gato en este universo </h2>
4 </div>
```

El bloque anterior asigna la variable “opcion” a la directiva “ng-switch” eso significa que si el valor de esta variable es “a” se mostrara el primer titular , por el contrario, si el valor es “b” se mostrara únicamente el segundo.

Por ahora, cuando ejecutas la aplicación, la variable “opcion” no tiene ningún valor, así que, al no calzar con los valores que se evalúan, no se muestra ninguno de los bloques de texto.

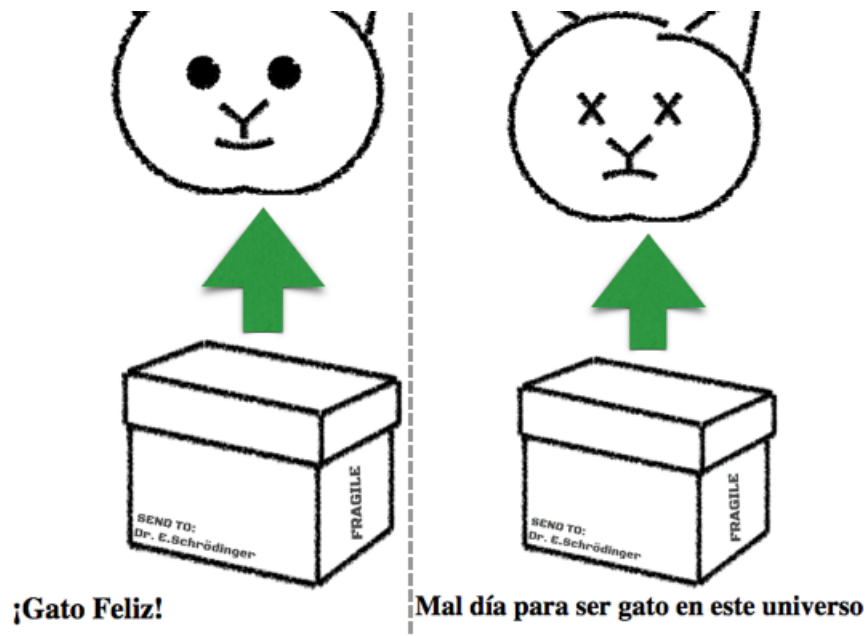
¡Es momento de hacer que estos valores se activen según las acciones del usuario! Primero, localiza el método “\$scope.opcionA”, allí dentro ya tenemos el código que define la imagen, agrega una línea extra para asignarle un valor a la variable “opcion”

```
1 $scope.opcion = 'a'
```

Ahora, repite el proceso en el controlador “\$scope.opcionB” pero esta vez, asigna el valor “b” a la variable “opcion”

```
1 $scope.opcion = 'b'
```

Una vez que tengas definidas ambas variables, al hacer click en los botones de opciones, se mostrara un texto diferente de acuerdo a los valores que active cada método.



Tenemos terminada una aplicación que controla la apariencia de acuerdo a las acciones del usuario, y lo mejor: ¡Ningún gato fue dañado en la creación de este capítulo! ;)

Eventos e interacción

Virtualmente todas las aplicaciones requieren de algún grado de interacción, desde insertar tu clave para entrar al sistema, hasta presionar el botón de pago en un carrito de compras virtual, el ordenador siempre está pendiente de todas tus acciones y las puede utilizar para modificar los elementos o contenidos de una aplicación.

Cualquier acción del usuario que pueda ser reconocida de alguna forma por el ordenador se entiende como un evento, con AngularJS podemos discriminar muchos diferentes eventos y relacionarlos a diferentes conductas. En este capítulo vamos a aprender a escuchar las principales acciones del usuario y las enlazaremos a través de AngularJS con las acciones de una app.

Clicks

El click es sin duda el evento de uso más común en una app, es tan importante y generalizado, de hecho ¡Estoy seguro que tuviste que hacer unos cuantos clicks para leer este libro!

Te voy a ahorrar tiempo detallando un evento que de seguro conoces a la perfección, mejor pongamos manos a la obra y hagamos un ejemplo para administrar clicks en AngularJS.

Comenzaremos con un código base que contiene una aplicación básica

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Eventos AngularJS</title>
6      <meta name="viewport" content="width=device-width">
7      <script type="text/javascript" src="angular.min.js"></script>
8  </head>
9  <body ng-app="eventosApp">
10     <div ng-controller="eventosController" id="principal">
11     </div>
12
13     <script type="text/javascript">
14
15         var miApp = angular.module('eventosApp', []);
16
17
```

```
18
19     miApp.controller('eventosController', function($scope) {
20         $scope.conteoClicks = 0;
21     })
22     </script>
23
24 </body>
25 </html>
```

En este documento estoy usando una aplicación llamada “eventosApp” tal como se define en la etiqueta `<body>` y también he incluido un controlador llamado “eventosController” dentro de un `<div>`. Unas líneas más adelante, en el código JavaScript encontrarás la variable “miApp” en la cual declaramos la aplicación. Adicionalmente tenemos el código del controlador “eventosController” y por ahora el controlador solo tiene una instrucción definiendo la variable “\$scope.conteoClicks” con el valor cero.

¡Estamos listos para poner manos a la obra! comenzaremos usando el elemento que se utiliza para escuchar clicks por excelencia en cualquier aplicación: un botón.

Aquí, usaremos la versión HTML5 de este componente, al incluir esta línea dentro del `<div>` que se encuentra en el código actual

```
1 <button> Click </button>
```

Ahora tenemos un botón en la app y al estar dentro del `<div>`, lo podemos administrar a través del controlador que tiene asignado.

Si queremos que un evento tenga una repercusión sobre la conducta de la aplicación, la forma más eficiente es crear una función que se ejecute cada vez que se active dicho evento. En AngularJS ya que los controladores son los encargados de gestionar el HTML y por ende sus eventos, usaremos una función dentro del controlador o un método, como se le conoce a este tipo de elementos.

El método que crearemos se llamará “saludo” y la sintaxis para crearlo será usando una variable, esta variable la igualaremos a una función

```
1 $scope.saludo = function(){
2     $scope.conteoClicks++
3 }
```

Recuerda usar siempre el objeto `$scope` para que los métodos sean accesibles desde el HTML.

Vamos a relacionar el botón con el método que acabamos de crear, utilizaremos la directiva “ng-click” y le asignaremos como valor el nombre del método que acabamos de crear, el código del botón quedaría de esta forma

```
1 <button ng-click="saludo()"> Click </button>
```

Como puedes notar, a la hora de invocar el método no estoy utilizando el objeto `$scope` porque desde el HTML no es necesario, también estoy invocando el nombre del método junto a paréntesis: `saludo()`, de esta forma el sistema puede comprender que estamos invocando un método y ejecutará el código que se encuentra en él.

Aunque la aplicación ya puede funcionar en este momento, nos falta un paso adicional para visualizar los datos, vamos a incluir una expresión que nos muestre un conteo de todas las veces que AngularJS detecte el evento click. Agrega esta línea de código justo después del botón

```
1 <strong> Cantidad de clicks: {{conteoClicks}}</strong>
```

Guarda tu documento y pruébalo en tu navegador, tendrás una aplicación que muestra un conteo de clicks similar a este:

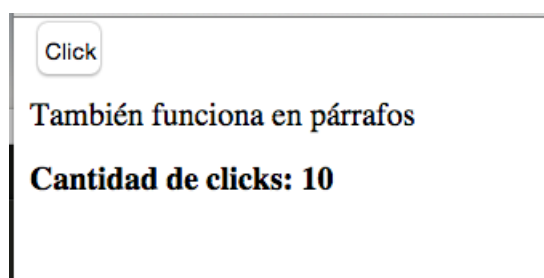


No esta mal, pero cualquiera puede hacer que un botón escuche un click, para eso están hechos después de todo ¿no?, que tal si lo probamos desde otro elemento, por ejemplo ,un párrafo.

En esta línea de código, vamos a repetir el mismo principio que acabamos de utilizar en el botón: le agregamos la directiva `ng-click` y asignamos el valor del método `saludo()`, la única diferencia será que esta vez lo haremos en una etiqueta de párrafo `<p>`

```
1 <p ng-click="saludo()"> También funciona en párrafos </p>
```

Ahora podemos detectar los clicks no solo del botón, sino también los que se realicen en el párrafo y gracias a que ambos elementos usan el mismo método ¡podemos llevar un conteo global de los clicks!



Finalmente, para que observes lo fácil que es manipular los eventos en AngularJS vamos a agregar un nuevo evento: el doble click.

En este caso usaremos un nuevo botón con un código muy similar al anterior, pero para escuchar el nuevo evento usaremos la directiva “ng-dblclick”. Agrega esta línea de código después del párrafo del ejemplo anterior

```
1 <button ng-dblclick="saludo()"> Doble Click </button>
```

Guarda el documento y pruébalo en tu navegador, notarás que al hacer un click normal en un botón y en el párrafo puedes aumentar el conteo , pero que en este último botón solo activarás el método “saludo()” al hacer doble click.



Eventos del mouse

En la sección anterior abarcamos el click, uno de los eventos más comunes, bueno, al menos de los eventos que generas con los botones del ratón porque nos quedan algunos eventos más por estudiar, que también puedes activar sin necesidad de hacer click.

Este periférico puede generar varios eventos, pero acá estudiaremos dos de los más comunes: “Mouse Enter” y “Mouse Leave”, el primero se activa cuando el puntero del ratón se posa sobre algún elemento, el segundo es su contra parte y se activa cuando él mismo se retira de un elemento.

Vamos a probar este evento creando primero un elemento visual para que te quede más claro su funcionamiento. Agrega estas líneas de código dentro de la etiqueta <head>

```
1 <style type="text/css">
2     #cuadrado{ background: red; width: 100px; height: 100px; margin: 30px 10\
3     px; }
4 </style>
```

Esto genera algunas propiedades visuales en el HTML, vamos a agregar un elemento para que sean visibles, agrega este <div> en tu código:

```
1 <div id="cuadrado" > </div>
```

Finalmente vamos a agregar una expresión que nos servirá para detectar el evento que active tu mouse

```
1 <strong>Estado: {{estado}}</strong>
```

El documento por ahora tiene una caja de color rojo de 100x100px la cual usaremos para hacer nuestras pruebas



Vamos ahora a agregar algunas líneas de programación dentro del controlador “eventos-Controller”, comenzaremos incluyendo la variable “\$scope.estado” con un valor inicial para desplegar en el documento antes de que se activen los eventos que vamos a evaluar.

```
1 $scope.estado = 'zzzzz';
```

El siguiente paso, es agregar métodos para administrar los diferentes eventos que vamos a soportar en esta aplicación. En este caso usaremos dos de ellos, el primero administrará la aplicación cuando el puntero del mouse se encuentre sobre un elemento

```
1 $scope.dentroDelDiv= function(){  
2     $scope.estado = 'MouseEnter activo';  
3 }
```

El segundo método que agregaremos es prácticamente igual al anterior, la diferencia será básicamente que está dirigido a administrar el evento que se activa cuando el puntero del mouse se encuentre fuera de un elemento


```
1 $scope.fueraDelDiv = function(){
2     $scope.estado = 'MouseOver activo';
3 }
```

Ahora que ya tenemos los métodos listos, vamos a asignarlos en el documento. Como pudiste observar en el ejemplo de los clicks, para asignar un evento, solo debes incluir la directiva correspondiente. En este ejemplo tenemos 2 eventos: el primero es “ng-mouseenter”, que sirve para detectar que el mouse esta dentro de un elemento, el segundo, “ng-mouseleave” detecta que el puntero del mouse está fuera del elemento.

En AngularJS puedes agregar un número indefinido de eventos dentro de cualquier elemento, siempre y cuando no se repita el mismo elemento. Así por ejemplo, el nuevo código del <div> con el cuadro rojo será:

```
1 <div id="cuadrado" ng-mouseenter="dentroDelDiv()" ng-mouseleave="fueraDelDiv()">
2 </div>
```

Este <div> al contener dos directivas puede leer dos diferentes eventos y al ser distintos eventos no hay ningún problema de compatibilidad.

Guarda el documento y prueba poner el puntero del mouse sobre el cuadrado rojo y luego retirarlo, podrás notar como AngularJS detecta cada evento.



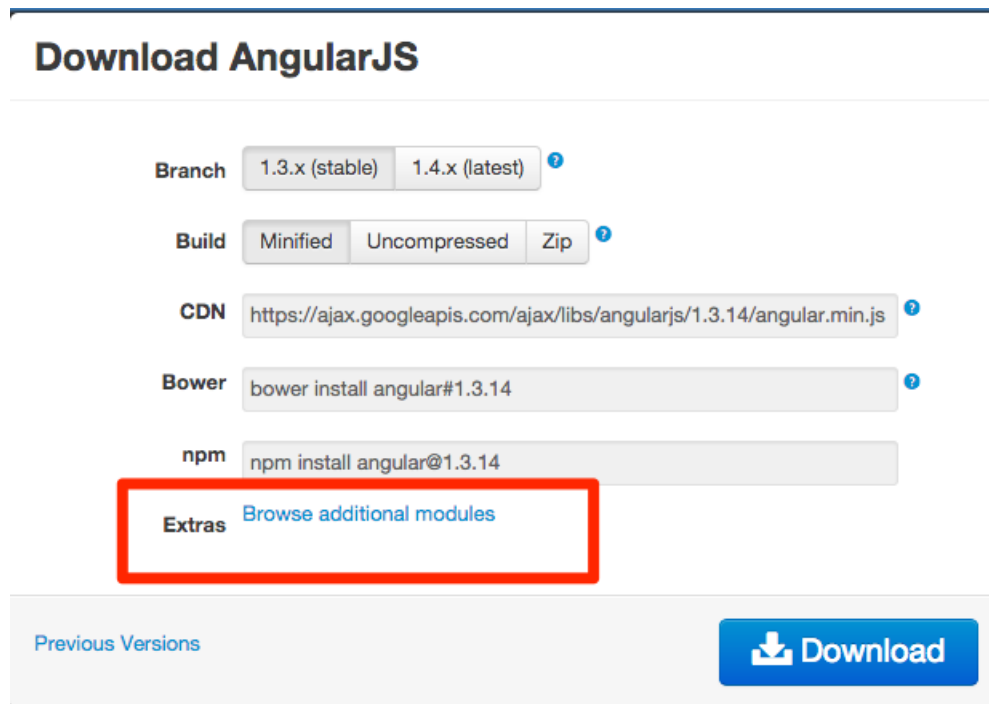
Eventos para móviles

Al trabajar en dispositivos móviles encontrarás algunos cambios importantes, por ejemplo, al utilizar una pantalla táctil aparecen nuevos eventos que no están disponibles en un ordenador de escritorio.

Vamos a probar los eventos específicos para móviles usando el ejemplo anterior, pero agregando algunas adaptaciones para este nuevo contexto.

AngularJS no tiene soporte nativo para los eventos táctiles propios de un dispositivo móvil, pero es sumamente sencillo habilitarlos, lo único que debes hacer es incluir el módulo “angular-touch” dentro de tu aplicación.

Puedes encontrar la dirección de esta librería entrando al sitio oficial de AngularJS y presionar el botón “Download” , allí encontraras la ventana con las opciones de descarga, busca el apartado “Extras” y presiona el enlace “Browse additional modules”



Esto abrirá un enlace donde verás todos los módulos adicionales disponibles, localiza el módulo “angular-touch.min.js”

```

../
docs/
i18n/
angular-1.3.14.zip
angular-animate.js
angular-animate.min.js
angular-animate.min.js.map
angular-aria.js
angular-aria.min.js
angular-aria.min.js.map
angular-cookies.js
angular-cookies.min.js
angular-cookies.min.js.map
angular-csp.css
angular-loader.js
angular-loader.min.js
angular-loader.min.js.map
angular-messages.js
angular-messages.min.js
angular-messages.min.js.map
angular-mocks.js
angular-resource.js
angular-resource.min.js
angular-resource.min.js.map
angular-route.js
angular-route.min.js
angular-route.min.js.map
angular-sanitize.js
angular-sanitize.min.js
angular-sanitize.min.js.map
angular-scenario.js
angular-touch.js
angular-touch.min.js
angular-touch.min.js.map
angular.js
angular.min.js
angular.min.js.map
errors.json
version.json
version.txt

```

Al abrirlo verás el código fuente de este módulo, copia y guarda el URL. Para instalar este módulo en un documento solo tienes que incluir esta línea de código:

```

1 <script type="text/javascript" src="https://code.angularjs.org/1.3.14/angular-to\
2 uch.min.js"></script>

```

En este caso hemos usado la versión 1.3 de AngularJS, recuerda usar el URL que acabas de copiar en el atributo “src” para reemplazarlo por la versión más actualizada.

Toma en cuenta que siempre debes poner esta línea DESPUÉS de la invocación a la librería de AngularJS, de lo contrario puedes tener errores en tu documento.

Ahora que tenemos instalado correctamente el nuevo componente en el documento, es necesario inyectarlo dentro de AngularJS, para eso tienes que agregar el valor “ngTouch” dentro de la invocación del módulo de la app de esta forma:

```

1 var miApp = angular.module('eventosApp',['ngTouch']);

```

¡Perfecto! ahora tu aplicación ya tiene instalado y configurado el soporte para eventos táctiles.

Para este ejercicio vamos a darle soporte al evento “swipe” que es la acción de presionar la pantalla y arrastrar hacia alguna dirección, es el gesto clásico que usas para pasar de pagina en un libro. En un dispositivo móvil tienes 4 versiones de este evento según la dirección en la que arrastres el índice: izquierda, derecha, arriba o abajo. AngularJS le da soporte a todos ellos pero para mantener breve este ejercicio usaremos solo dos de ellos: izquierda y derecha.

Para soportar estos eventos agregaremos dos nuevos métodos dentro del controlador de la aplicación, el primero lo usaremos para administrar el evento “swipe” hacia la izquierda y la acción que tendrá es la de modificar el valor de la variable “\$scope.estado”

```
1 $scope.swipeIzq = function(){
2     $scope.estado = '<-- Swipe ';
3 }
```

El segundo método tiene un rol similar, pero se activará al hacer “swipe” hacia la derecha y asignará otro valor para la misma variable

```
1 $scope.swipeDe = function(){
2     $scope.estado = 'Swipe --> ';
3 }
```

Solo resta asignar los nuevos eventos dentro del HTML, usaremos el mismo ejemplo del ejercicio anterior y solo le agregaremos el soporte para los nuevos eventos, en este caso “ng-swipe-left” activará el método “swipeIzq()” cada vez que se haga un “swipe” en dirección derecha dentro del contenedor, mientras que “ng-swipe-right” activa el método “swipeDe()” cuando se ejecuta esa acción hacia la derecha.

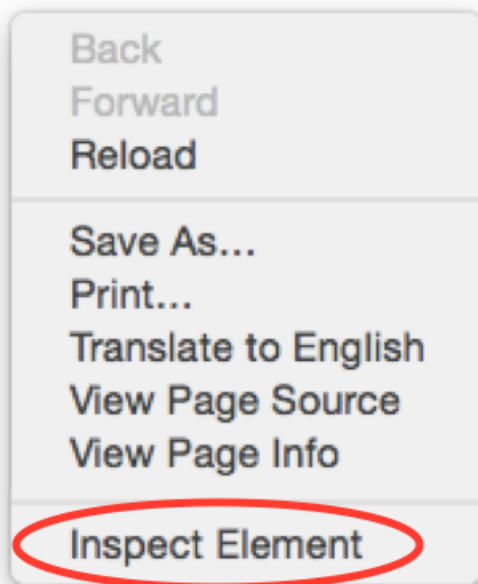
```
1 <div id="cuadrado" ng-mouseenter="dentroDelDiv()" ng-mouseleave="fueraDelDiv()" \
2   ng-swipe-left="swipeIzq()" ng-swipe-right="swipeDe()">
3 </div>
```

Tu código esta listo, pero hay un detalle adicional para poder probar tu código correctamente: necesitas una pantalla táctil o un simulador. Vamos a aprender como emular un sitio móvil usando Chrome

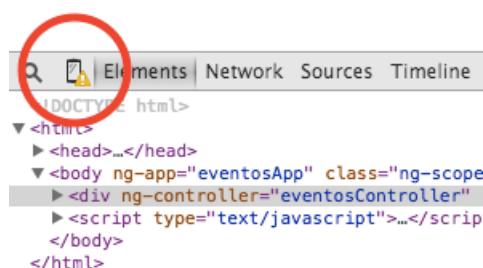
Simulando una pantalla táctil en Chrome

Para probar los eventos táctiles, necesitas una pantalla con soporte **táctil**, de lo contrario no puedes probar correctamente los eventos que provienen de allí, afortunadamente Google Chrome tiene un muy buen simulador, muy práctico para estos casos.

Para activarlo solo tienes que abrir el documento en tu navegador y en cualquier parte del mismo hacer click derecho, allí te aparecerá un menú contextual, busca la opción “Inspeccionar Elemento” o “Inspect Element” de acuerdo a los ajustes de tu navegador:



Esta opción despliega el menú de depuración de Chrome, allí debes seleccionar la opción para emulación móvil haciendo click en el icono de móvil que aparece de segundo al lado izquierdo



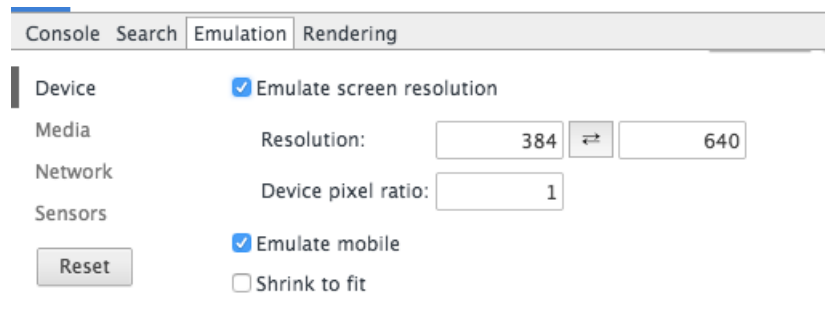
Ahora puedes ver tu documento como se desplegaría en un dispositivo móvil, por ejemplo puedes definir que equipo desees emular, busca el menú “device” que se encuentra en la parte superior izquierda, solo para elegir un equipo muy popular en el que posiblemente varios usuarios vean el documento escogeré el Nexus 4, pero tu puedes escoger el que desees.



También puedes seleccionar más opciones de resolución y tamaño, seleccionando el menú de opciones en la parte inferior derecha

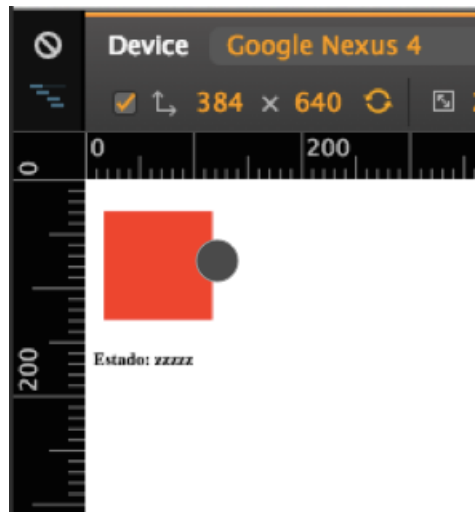


Para asegurarte que todo se vea correctamente asegúrate de desmarcar la opción “Shrink to Fit” para que el contenido no se escale en tu navegador

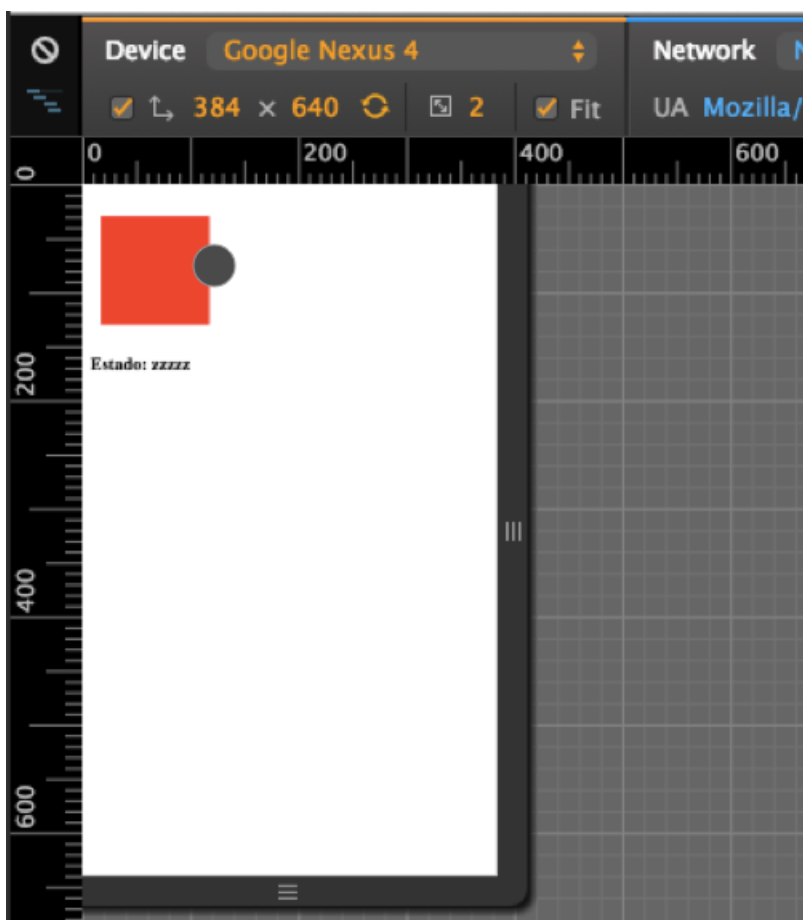


Allí también encontrarás otras opciones para emular como la resolución , tamaño y orientación de pantalla, te invito a explorar las opciones de esta herramienta, pero por ahora vamos a enfocarnos en el ejercicio que estamos haciendo.

Si te fijas, cuando tienes activa la vista de emulación, desaparece el puntero normal del mouse y aparece un círculo, este círculo representa un dedo y además de darte una idea del espacio que ocupa normalmente el mismo, te indica que los eventos táctiles están activos, de hecho, si pasas este nuevo puntero del mouse sobre el cuadro rojo, notarás que ya no se activan los eventos “mouseenter” o “mouseleave” porque estos eventos no existen en los dispositivos móviles



Vamos a probar los nuevos eventos en nuestra aplicación: pon el puntero del mouse al medio del contenedor, presiona el click del mouse y arrastra hacia la derecha, así simularas el evento “swipe” y eso activará el mensaje dentro de la aplicación, repite el proceso pero arrastra el mouse esta vez a la derecha y veras como se activa el segundo método.



Ya le estas dando soporte a eventos móviles y de paso aprendiste a usar una herramienta ¡Estoy seguro que este truco te será de utilidad en muchos otros casos!

Eventos de teclado

AngularJS tiene una amplia gama de eventos soportados y no quisiera que pienses que solamente usamos elementos relacionados con el mouse o pantallas táctiles, después del mouse el periférico que más utilizas para enviar información al tu equipo es el teclado y no quisiera terminar este capítulo sin antes mostrarte algunos eventos relacionados con el.

Como ya vimos en los ejemplos anteriores AngularJS utiliza las directivas para asignar eventos y sus respectivos controladores, así que vamos a agregar directamente un nuevo elemento a nuestro documento con estos elementos incluidos. Copia este código dentro del <div> que usamos para los ejercicios de este capítulo:

```
1 <input type="text" ng-keydown="detectarTecla($event)" autofocus="true">
```


En este caso tenemos una etiqueta `<input>` para recibir texto, le incluí la directiva “ng-keydown” que se activa cada vez que presiones una tecla mientras tengas seleccionado este `<input>`, dicha directiva esta relacionado con un método llamado “detectarTecla()” y en este caso, a diferencia de los ejemplos anteriores estoy enviándole un parámetro con el valor “\$event”. Esta es una propiedad especial de AngularJS que me permite acceder los diferentes valores y propiedades relativos al evento en si, en este caso lo uso porque “ng-keydown” se activa indiscriminadamente cuando el usuario presiona una tecla, mientras que “\$event” al enviarme los datos detallados del evento, me sirve para conocer cual tecla en particular fue presionada.

Vamos ahora a incluir el nuevo método “detectarTecla()” en el controlador del documento.

```
1 $scope.detectarTecla = function(e) {  
2  
3 }
```

Como puedes notar dentro de la función estoy incluyendo el parámetro “e” que representa la información que recibe del objeto “\$event”.

dentro de los valores que devuelve dicho objeto, necesito uno en particular llamado “keyCode”, cada tecla tiene un código asignado en el sistema y allí se incluye el código de la tecla presionada.

Para hacer más interesante este ejercicio no usaremos las letras, sino que trabajaremos con las flechas del teclado, un elemento muy importante que generalmente pasa por alto y te puede ser muy útil en ocasiones

Las flechas del teclado tienen asignados los códigos del 37, al 40, así que vamos a usar una condicional switch para procesar cada valor y devolver un resultado según la flecha presionada, agrega este condigo dentro del método “\$scope.detectarTecla “

```
1 switch (e.keyCode) {  
2     case 37:  
3         $scope.estado = 'izquierda';  
4         break;  
5     case 38:  
6         $scope.estado = 'arriba';  
7         break;  
8     case 39:  
9         $scope.estado = 'derecha';  
10        break;  
11     case 40:  
12        $scope.estado = 'abajo';  
13        break;  
14 }
```

Como puedes notar, estamos procesando el valor del teclado que nos llega a partir de “e.keyCode” y le asignamos diferentes valores a la variable “\$scope.estado” según la tecla presionada.

¡Estamos listos para probar el resultado! guarda los cambios y prueba el documento en tu navegador, ahora cada vez que presiones una flecha la aplicación te responderá un mensaje con la dirección de la misma.

arriba

SECCION 3: MANIPULACION DE DATOS

Representar colecciones de datos

En una aplicación web, se manejan datos... ¡muchos datos! es casi seguro que en algún punto vas a tener que representarlos de manera estructurada, en AngularJS puedes crear listas a partir de estos datos de manera rápida y sencilla. En este capítulo te tengo un ejercicio que explora varios de los posibles escenarios que puedes encontrar cuando desarrolles tus apps.

Comenzaremos con un código que ya incluye algunas funcionalidades básicas para que ahorremos tiempo en detalles y podamos ir directamente al grano.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Colecciones de datos</title>
6      <script src="angular.min.js"></script>
7      <style>
8      </style>
9  </head>
10 <body ng-app="coleccionesApp">
11     <div ng-controller="controladorColecciones">
12
13         <ul>
14             <li>Items</li>
15             <li>Items</li>
16         </ul>
17
18     </div>
19
20     <script>
21         var coleccionDatosApp = angular.module('coleccionesApp', []);
22
23         coleccionDatosApp.controller('controladorColecciones', function ($scope)\
24     {
25
26         $scope.datos = ['piedra', 'papel', 'tijera'];
27
28
29         $scope.agregar = function(){
```

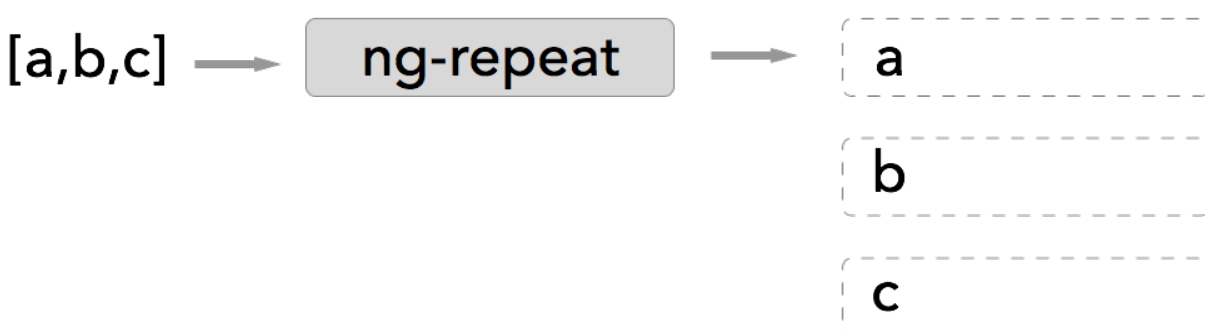
```
30
31     }
32
33     $scope.mostrarDatos = function(info){
34
35     }
36
37     })
38 </script>
39 </body>
40 </html>
```

En este código base ya tengo instalada y configurada una aplicación AngularJS, también he creado un controlador llamado “controladorColecciones” al que le he incluido 2 métodos, por ahora vacíos: “agregar” y “mostrarDatos”. Finalmente tenemos un arreglo llamado “datos” con un grupo de valores, precisamente, hablaremos en detalle de este elemento a continuación.

Crear listas con arreglos

Los arreglos son un tipo de dato de uso muy frecuente en las aplicaciones, se utilizan generalmente para almacenar grupos de valores, AngularJS tiene una directiva llamada “ng-repeat” especializada en procesar este tipo de datos.

El funcionamiento de esta directiva es iterar sobre un valor complejo y repetir bloque en el que se encuentra definida. Por ejemplo, si tenemos un arreglo de 3 valores y lo procesamos a través de un “ng-repeat”, el resultado serán tres bloques, cada uno de ellos representando uno de los valores del arreglo.



La directiva “ng-repeat” utiliza una sintaxis especial para iterar o analizar cada elemento de un arreglo, el esquema general de la sintaxis es:

```
1 ng-repeat=" NOMBRE_DE_INSTANCIA in ARREGLO"
```

El primer valor, es el nombre de una instancia que almacenará dinámicamente cada uno de los valores del arreglo, esta instancia cambia de valor en cada iteración, lo que nos permite desplegar y procesar los valores en cada uno de los ítems que se van a generar. Luego, tenemos el comando “in” que permite iterar según el número de elementos que vamos a evaluar y finalmente el “ARREGLO” donde debes incluir el nombre del arreglo que será procesado.

¡Vamos a aplicar estos conceptos en un ejemplo práctico! En el código base de esta aplicación ya tenemos definido un arreglo llamado “datos”

```
1 $scope.datos = ['piedra', 'papel', 'tijera'];
```

Como puedes ver, tenemos allí almacenados tres valores, para representarlos en una lista vamos a utilizar la directiva “ng-repeat” dentro de una etiqueta

```
1 <li ng-repeat="datoIndividual in datos"> {{datoIndividual}}</li>
```

En este caso “datoIndividual” será la instancia que almacenará cada uno de los valores que procesaremos y “datos” es el nombre del arreglo que contiene dichos valores. Como puedes notar, también he incluido una expresión representando el valor de la instancia, esto permite representar en el HTML cada uno de los valores del arreglo

Una vez que ejecutes el código, “ng-repeat” repetirá el bloque por cada uno de los valores de “datos”, en este caso tendremos una lista similar a esta

- piedra
- papel
- tijera

¡Eso es todo! acabamos de procesar un arreglo con una sola línea de código

¿Fácil, no?

Vamos a hacer este ejercicio aun más interesante usando algo de interacción.

Agregar elementos a una lista

Vamos a incluir nuevos elementos a la lista en tiempo de ejecución, para ello, vamos a incluir un par de líneas extra de código, copia este texto justo después de la lista

```
1 <input type="text" ng-model="nuevoItem">
2 <button ng-click="agregar()" > Guardar </button>
```

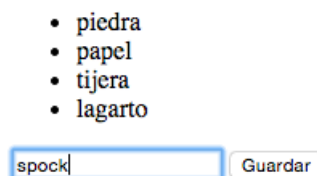
Esto genera una entrada de texto con un botón, que nos permitirá inyectar valores a la lista. Observa que en el `<input>` he incluido una directiva “ng-model” que crea un modelo o un contenedor de valores dentro del sistema, en este caso, llamaremos a este modelo “nuevoItem”.

También, en el botón tenemos la directiva “ng-click” que relaciona el evento click de este botón con un método llamado “agregar()”. Este método ya se encuentra en el código base, pero debemos agregar una línea extra, para que quede similar a esto:

```
1 $scope.agregar = function(){
2     $scope.datos.push( $scope.nuevoItem );
3 }
```

El comando que incluimos toma el arreglo “datos” y a través del comando de JavaScript llamado “push” le inyecta los valores del modelo. En otras palabras, al hacer click, AngularJS tomara el valor que se encuentre en la entrada de texto y lo incluirá en la lista

Guarda el documento, agrega un texto en la casilla y al presionar el botón ¡Verás como los nuevos elementos se agregan de inmediato!



Toma en cuenta que estos elementos nuevos se están asignando en memoria y en tiempo de ejecución solamente, o sea que al refrescar el documento tendrás únicamente los valores originales.

Tenemos una aplicación que guarda ítems en tiempo real... ¿Ya sientes el poder de AngularJS? pues prepárate, porque ¡aun no terminamos!

Asignar eventos en listas dinámicas

Muy probablemente cuando apliques esta técnica para desplegar listas, tardarás solo unos minutos para descubrir que te falta por aprender algo.. ¿que pasa si necesitas hacer listas más complejas? por ejemplo, una lista que tenga soporte para eventos.

Asignar eventos dentro de los ítems de una lista es relativamente fácil, solo necesitas agregar la directiva “ng-click” en conjunto con el “ng-repeat” para que todos los elementos de la lista puedan activar conductas basadas en un botón.

Sin embargo, al repetir el evento, también repites la conducta, o sea, todos los ítems de la lista ejecutarán exactamente la misma acción, esto puede ser útil en algunos casos, pero ¿Como hacer para que cada ítem se comporte distinto? Para esto, usaremos datos dinámicos.

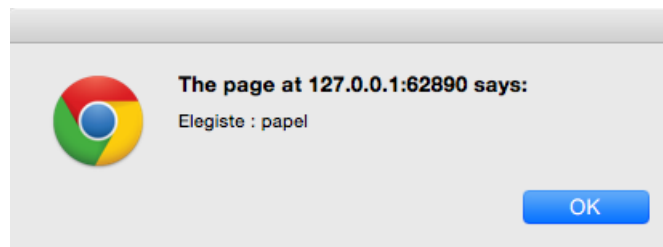
Los métodos pueden escuchar parámetros que envían información dinámica, para incluir esta información dinámica podemos tomar mano de las expresiones, por ejemplo, vamos a modificar el “ng-repeat” del ejercicio actual y le incluiremos una directiva “ng-click”

```
1 <li ng-repeat="datoIndividual in datos" ng-click="mostrarDatos(datoIndividual)">\n2   {{datoIndividual}}</li>
```

Como puedes ver, la directiva “ng-click” en este caso usa el método “mostrarDatos” y allí recibe los datos contenidos en “datoIndividual”. Como recordarás, esta propiedad contiene un valor dinámico, que se asignará según el valor correspondiente a cada uno de los elementos del arreglo.

```
1 $scope.mostrarDatos = function(info){\n2     alert('Elegiste : ' + info)\n3 }
```

Ahora, al hacer click sobre cada ítem obtendrás una ventana con la información respectiva.



Ya puedes mostrar una lista y asignarle eventos con datos dinámico... ¡Nada mal! pero es momento que llevemos este ejercicio al ¡Nivel Épico!

Crear listas con objetos

Los arreglos pueden contener muchos tipos de datos, hasta ahora hemos usado cadenas de texto simples, pero también puedes usar objetos para representar datos más complejos.

Es muy posible que cuando desarrolles tus aplicaciones en el mundo real te encuentres con situaciones que van más allá de un simple listado de elementos, sino de representar valores complejos con múltiples datos y para cerrar este capítulo te voy a mostrar como hacerlo.

Comencemos agregando un arreglo de objetos dentro de la aplicación. Justo después del arreglo “datos” agrega este código


```
1 $scope.heroes = [  
2   { nombre:"Arnold", apellido:"Schwarzenegger"},  
3   { nombre:"Chuck", apellido:"Norris"},  
4   { nombre:"Bruce", apellido:"Lee"}  
5 ]
```

Acá, tenemos un arreglo, que a su vez contiene tres objetos, cada uno de ellos tiene a su vez contiene dos valores: “nombre” y “apellido”.

Ahora que tenemos este objeto complejo disponible, vamos a agregar una nueva lista en el HTML, copia este código en el documento, después del ejercicio de la lista anterior

```
1 <ul>  
2   <li ng-repeat="hero in heroes"> {{hero.nombre}}</li>  
3 </ul>
```

Notarás que la sintaxis es muy similar al ejemplo anterior, tenemos el “ng-repeat” usando instancias y procesando el arreglo “heroes”, pero la principal diferencia es que en el ejemplo anterior la instancia almacenaba un valor simple, una cadena de texto, en este caso , la instancia almacena un objeto, por esta razón la expresión no solo dice “hero” sino que usa una de las propiedades que se encuentran en los objetos, en este caso “nombre”

El resultado, es una lista con un número de ítems igual al de los valores del arreglo que muestra la propiedad “nombre” en cada uno de ellos

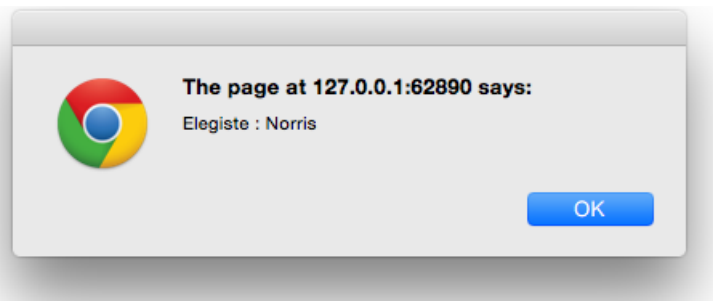
- Arnold
- Chuck
- Bruce

En una lista con estos valores, definitivamente tenemos que agregarle algo de acción, así que vamos a incluirle, la directiva “ng-click”, pero a diferencia del ejercicio anterior enviaremos como dato dinámico otra de las propiedades del objeto, en este caso el “apellido”.

```
1 <li ng-repeat="hero in heroes" ng-click="mostrarDatos(hero.apellido)"> {{hero\  
2 .nombre}}</li>
```

Ahora al hacer click, invocaremos el mismo método que usamos en el ejemplo anterior para mostrar los datos de cada ítem, pero esta vez, estaremos usando dos propiedades diferentes, una para representar en la lista y otra para el método.

- Arnold
- Chuck
- Bruce



Tenemos terminada una lista usando iteraciones con objetos y asignando valores dinámicos con sus propiedades

¡Chuck Norris aprueba este ejercicio!

Acceder datos externos

Una de las operaciones más frecuentes en una aplicación web será importar datos externos, esta técnica te permite escalar tu aplicación, insertar información dinámica y hasta conectarte con API's externas.

En este capítulo aprenderemos a utilizar uno de los servicios nativos de AngularJS para importar un archivo externo, leer sus contenidos y desplegar los resultados dentro de una aplicación.

Arranquemos este ejercicio con un código base que contiene los elementos básicos para comenzar a trabajar.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Datos</title>
6      <script src="angular.min.js"></script>
7      <style>
8      </style>
9  </head>
10 <body ng-app="datosApp">
11     <div ng-controller="controladorDatos">
12
13         <h1>A-Team</h1>
14
15         <p></p>
16
17         <ul>
18             <li></li>
19         </ul>
20
21     </div>
22     <script>
23         var misDatos = angular.module('datosApp', []);
24
25         misDatos.controller('controladorDatos', function ($scope) {
26
27         })
```

```
28
29     </script>
30 </body>
31
32 </html>
```

En este documento tenemos funcionando una aplicación AngularJS con un controlador llamado “controladorDatos”, ya tenemos también los elementos HTML para desplegar los datos que vamos a cargar: un párrafo y una lista; por ahora ambos se encuentran vacíos pero pronto vamos a trabajar con ellos.

Guarda este archivo con el nombre “datos.html”. Por ahora, si pruebas este documento, solo verás un titular con un párrafo y una lista vacíos.

A-Team

.

Preparando los datos externos

Antes de continuar con este ejercicio, necesitamos preparar el archivo en el que almacenaremos los datos y posteriormente cargaremos en la aplicación.

Técnicamente AngularJS puede cargar cualquier tipo de archivo, sin embargo, en este ejercicio usaremos un documento en formato JSON ya que es uno de los más populares y muy posiblemente lo utilices en tus aplicaciones.

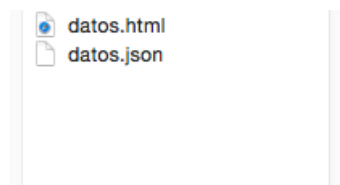
Vamos entonces a crear un archivo llamado “datos.json” en el cual incluiremos estos datos:

```
1 {
2   "integrantes": [
3     {
4       "nombre": "John 'Hannibal' Smith",
5       "especialidad": "Estrategia",
6       "rango": "Coronel"
7     },
8     {
9       "nombre": " Templeton Peck",
```

```
10         "especialidad": "Logística",
11         "rango": "Teniente"
12     },
13     {
14         "nombre": " H.M. Murdock",
15         "especialidad": "Piloto",
16         "rango": "Capitán"
17     },
18     {
19         "nombre": "B.A Baracus",
20         "especialidad": "Artillería",
21         "rango": "Sargento"
22     }
23 ],
24     "historia": "Hace 10 años un tribunal militar condenó a prisión a un grupo d\
25 e comandos por un crimen que no cometieron, esos hombres escaparon de prisión y \
26 se instalaron clandestinamente en Los Angeles... hoy, aunque el gobierno los busca\
27 , si alguien tiene un problema, necesita ayuda y puede localizarlos,tal vez pued\
28 a contratarlos"
29
30 }
```

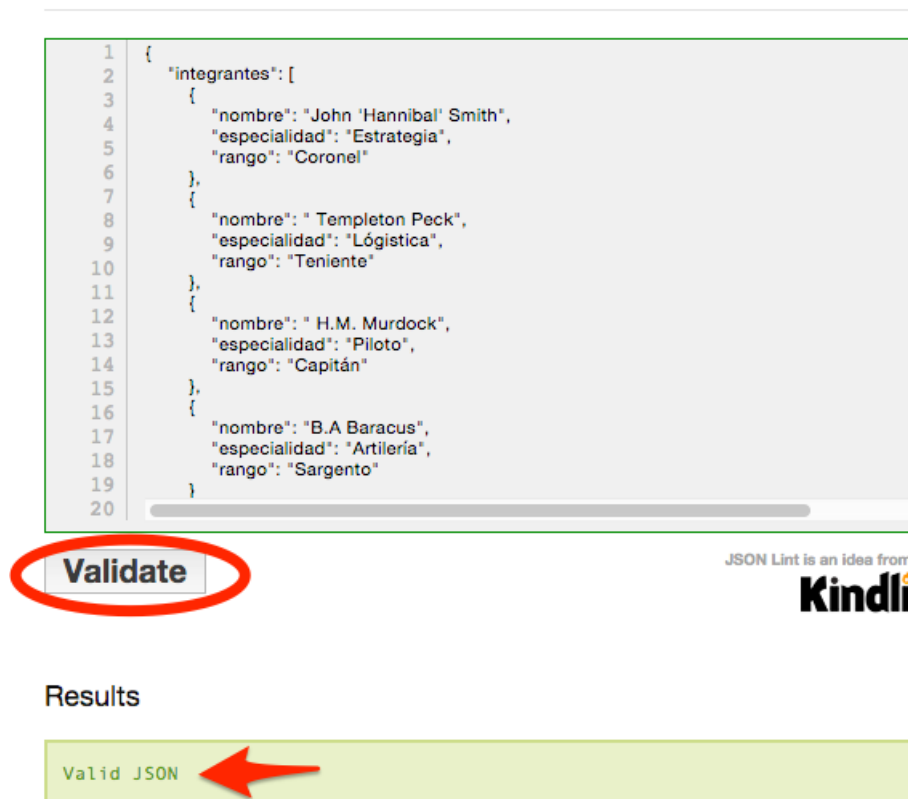
En este documento tenemos dos propiedades principales: “integrantes” en la que se encuentra almacenado un arreglo con 4 diferentes objetos e “historia” que incluye una cadena de texto. La estructura que estamos usando en este ejemplo tiene por objetivo mostrarte un documento con valores mixtos similares a los que posiblemente te encontrarás en el mundo real.

Procederemos a guardar el archivo, su ubicación, al menos para los efectos este ejercicio será la misma carpeta en la que se encuentre el archivo “datos.html”



Ahora que tienes listo el archivo en que se almacenará la información, quiero aprovechar para darte un consejo: es muy común que tengas errores de sintaxis con este tipo de archivos, en especial cuando comienza a dar tus primeros pasos. Si tu archivo de datos tiene cualquier error, toda la aplicación puede fallar, así que te recomiendo que siempre los valides para estar 100% seguro de que todo funcionará perfecto desde el inicio.

Puedes usar el servicio gratuito de jsonlint.com para validar tus documentos en línea, solo tienes que pegar el código y verificarlo.



El sistema te avisara si todo esta correcto o si hay algún error te marcará la línea para que lo soluciones de inmediato.

¡Trabajar con un código valido te ahorra tiempo y dolores de cabeza innecesarios!

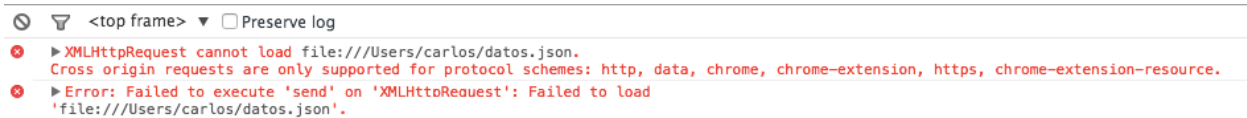
Consideraciones de seguridad y permisos de lectura

Otro aspecto importante que te debo mencionar antes de continuar este ejercicio es que tomes en cuenta que debido a la arquitectura de los navegadores modernos, debes probar el código desde un servidor local, de lo contrario muy posiblemente no te funcione correctamente.

Esto no se debe a que el AngularJS tenga problemas para cargar los datos, o que tu código tenga errores, sino a que los navegadores tienen restricciones de seguridad para cargar archivos externos de manera local, por ejemplo si al abrir tu ejemplo en la barra de dirección aparece al inicio este texto

1 file://

Significa que el archivo esta funcionando en modo local y en la mayoría de los navegadores la aplicación no funcionara reportando un error de XMLHttpRequest similar a este:



Para solucionarlo, tienes varias opciones, la más simple es probar en otro navegador, por ejemplo Safari o Internet Explorer que generalmente no tienen esta restricción de seguridad. Otra opción es instalar un servidor local como XAMPP o WAMPP y probar tus archivos desde allí, puedes encontrar estos programas en las siguientes direcciones:

- <https://www.apachefriends.org/index.html>
- <http://www.wampserver.com/en/>

Mi recomendación personal es que uses Adobe Brackets y utilices su vista previa que permite probar los documentos en un servidor local sin necesidad hacer ninguna instalación o configuración especial.

Toma en cuenta que una aplicación web esta pensada precisamente funcionar en un entorno web (valga la redundancia) este problema solamente se presentara en las condiciones de desarrollo, cuando ejecutes tu aplicación on line puedes tener total seguridad de que funcionara exactamente igual que en tu servidor local.

Cargar información desde servidores externos

En algunos casos especiales es posible que necesites cargar información desde servidores de terceros, esto, en algunas ocasiones puede detonar un error de permisos en los navegadores y activar un error de acceso a dominios externos similar a este



Este problema se soluciona fácilmente al definir una política de acceso a recursos externos o CORS (acrónimo de Cross Origin Resource Sharing) la configuración cambia según cada servidor y los detalles se salen del alcance de este libro pero puedes encontrar información completa para la mayoría de servidores y lenguajes en este sitio web

<http://enable-cors.org/>

Cargando un archivo externo

AngularJS tiene servicios dedicados que nos permiten ejecutar tareas complejas de manera fácil y rápida, por ejemplo, para cargar un archivo externo debemos hacer una petición HTTP y cargar asincrónicamente sus datos. En AngularJS solo necesitas usar el servicio `$http` que agrupa una serie de operaciones para interactuar con datos externos.

A manera de esquema general, la sintaxis de este servicio funciona de esta manera:

```
1 $http.TIPODELLAMADA('NOMBRE DE ARCHIVO')
```

El servicio `$http` utiliza diferentes tipos de llamadas para interactuar con los datos externos, por ejemplo, para enviar datos desde la aplicación hacia el exterior utiliza “POST”, mientras que para traer datos, usa “GET”.

Aplicando estos conceptos en un ejemplo específico, si quisiéramos invocar los datos de un archivo llamado “archivo.txt” usaríamos un código así

```
1 $http.get('archivo.txt')
```

Este comando tiene la capacidad de conectarse al documento ‘archivo.txt’ pero aun no tenemos acceso a procesar los datos que se encuentran allí.

Al ser esta, una llamada asincrónica debemos esperar a que el proceso de descarga finalice correctamente para tener acceso a los datos del documento. AngularJS nos ofrece dos métodos adicionales que nos ayudan a administrar el flujo de la información: “success” que se activa cuando la carga finalizo correctamente y “error” que se activa en caso de que surja algún problema.

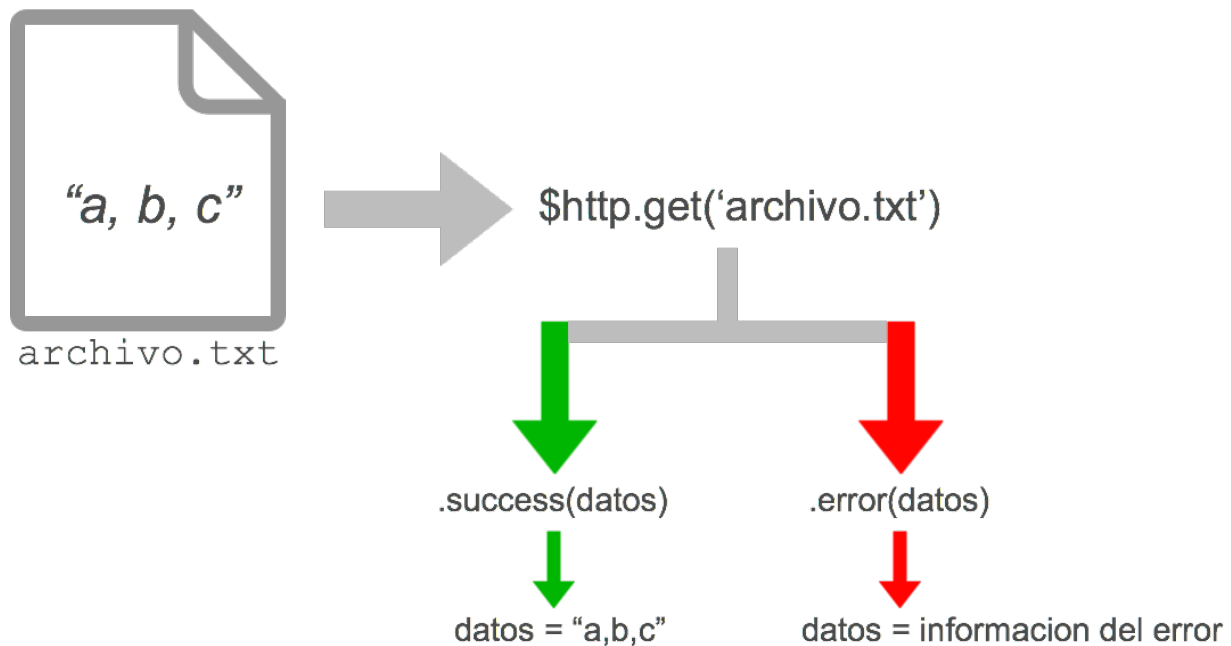
Continuando entonces el ejemplo de carga del documento “archivo.txt”, para administrar correctamente los datos deberíamos usar un código similar a este:

```
1 $http.get('archivo.txt')
2 .success( function(datos){
3
4 } )
5 .error( function(datos){
6
7 })
```

En este caso, si la la carga del archivo es exitosa, se ejecutara “success()” y los comandos que se encuentran allá, si por el contrario hay algún error en la carga, se ejecutaran los comandos que se encuentren en “error()”

Pon atención especial a que ambos elementos usan un parámetro llamado “datos”, allí se encuentra la información de respuesta para la invocación del archivo, si la carga es exitosa, allí se encuentran almacenados todos los datos del documento solicitado, si la carga tuvo errores, se almacenarán entonces los detalles del problema.

En este esquema puedes ver en detalle el proceso de carga que acabamos de realizar.



Suficiente con la teoría ¡es momento de entrar en acción! vamos a cargar y procesar un archivo externo, ahora que comprendes todos los detalles del proceso solo nos tomará unos minutos.

El primer paso es incluir el servicio \$http en el controlador en que lo vamos a utilizar, solo tienes que agregarlo en la llamada del controlador de la misma forma ya lo hacemos con el objeto \$scope

```
1 misDatos.controller('controladorDatos', function ($scope , $http)
```

Ahora que tenemos disponible el servicio \$http, podemos utilizar sus capacidades para cargar archivos externos en cualquier parte del controlador.

Estamos listos para cargar un archivo, en este caso lo haremos dentro de un metodo llamado "inicializar"

```
1 $scope.inicializar = function(){
2
3     $http.get('datos.json').
4         success(function(datos) {
5
6             $scope.equipo = datos;
7
8         });
9
10 }
```

Como estamos usando el servicio “\$http” y como vamos a descargar información usaremos “get”. Si todo carga correctamente se ejecutara “success” que en este caso recibe el parámetro “datos” donde se encuentra toda la información que acabamos de cargar.

Al igualar la variable “equipo” a “datos” estamos insertando dentro de la aplicación, toda la información recolectada del archivo externo.

Lo único que falta ahora es invocar este método, esto puedes hacerlo de varias formas, por ejemplo relacionando un evento “ng-click” para que los archivo se carguen al presionar un elemento en pantalla.

Sin embargo, esta vez te voy a mostrar un truco sencillo para que se carguen los datos justo el momento que arranca la aplicación, lo único que tienes que hacer es invocar el método en la línea siguiente después de crearlo.

```
1 $scope.inicializar()
```

Ahora cuando tu aplicación arranque va a ejecutar el método “inicializar()” inmediatamente después de haberlo creado.

Desplegar los datos cargados

Para este punto ya tenemos disponibles todos los datos del documento externo almacenados en la variable “equipo” solo resta desplegarlos en el la aplicación.

Recuerda que dentro del documento externo teníamos 2 propiedades : “historia” y “equipo”, vamos a comenzar a trabajar con esta primera propiedad, para desplegar sus datos debes modificar el párrafo que esta en el código, incluyendo esta expresión

```
1 <p> {{equipo.historia}} </p>
```

La siguiente propiedad, llamada “equipo”, contiene un arreglo con objetos, estos elementos los puedes procesar como cualquier otra lista, tal como examinamos en el capítulo anterior .

En este caso, crearemos una instancia llamada “integrante” e iteraremos sobre los elementos que componen el arreglo que se encuentra en “equipo.integrantes”

```
1 <ul>
2   <li ng-repeat="integrante in equipo.integrantes">
3     {{integrante.rango}} {{integrante.nombre}} ( {{integrante.especialidad}}\
4   )
5   </li>
6 </ul>
```

Hemos terminado así nuestra aplicación, guarda los cambios y al abrir el ejercicio veras como los datos del archivo externo se cargan automáticamente y se despliegan en la pantalla

A-Team

Hace 10 años un tribunal militar condenó a prisión a un grupo de comandos por un crimen que no cometieron, esos hombres escaparon de prisión y se instalaron clandestinamente en Los Angeles... hoy aunque el gobierno los busca, si alguien tiene un problema, necesita ayuda y puede localizarlos,tal vez pueda contratarlos

- Coronel John 'Hannibal' Smith (Estrategia)
- Teniente Templeton Peck (Lógica)
- Capitán H.M. Murdock (Piloto)
- Sargento B.A Baracus (Artillería)

Ahora que eres un experto en carga de datos, prueba modificar la información del documento externo, verás que al refrescar el navegador se reflejaran los cambios en la aplicación.

SECCION 4: EXPANDIENDO ANGULAR

Integrando AngularJS con Bootstrap

Bootstrap es un framework especializado para crear interfaces en aplicaciones web. Creado por Twitter a partir de sus propias necesidades y liberado bajo licencia de código abierto, Bootstrap - a diferencia de AngularJS - está totalmente enfocado en la parte gráfica de una aplicación.

Al utilizar Bootstrap usas un conjunto de clases CSS y algunas herramientas de apoyo en JavaScript para crear la interfaz gráfica de una aplicación. Este framework trabaja con una estructura responsiva que le permite adaptarse y desplegarse correctamente en cualquier dispositivo móvil.

El objetivo principal de Bootstrap es disminuir los tiempos de desarrollo eliminando tareas repetitivas, utilizando patrones y elementos de uso común, brindando herramientas consistentes y probadas para eliminar o reducir al máximo los procesos de diseño, depuración de una app.

Es importante aclarar que al crear una aplicación AngularJS no requieres de ninguna otra herramienta adicional, sin embargo, tomando en cuenta que AngularJS solo se encarga de la lógica de la aplicación, tarde o temprano necesitaras desarrollar una interfaz para la interacción con los usuarios. En este libro usaremos Bootstrap porque es una de las soluciones más populares y sencillas de utilizar. En caso de que prefieras trabajar con otras opciones, te sugiero algunas alternativas como:

- Foundation
- HTML5 Boilerplate
- YUI
- Skeleton

Todas ellas pueden funcionar igual de bien que Bootstrap para tu proyecto y gracias a que AngularJS es totalmente independiente de los elementos gráficos, no tendrás mayor problema para integrarlas en tu aplicación.

¿Por qué usar Bootstrap?

Bootstrap es uno de los frameworks más populares del mercado y además es completamente gratuito, lo que lo convierte en una excelente elección para trabajar tus proyectos basados en AngularJS. Entre las razones por las que tantos desarrolladores lo eligen tenemos:

- **Es un buen punto de partida:** Usa un CSS normalizado para que los elementos de tu documento se muestren de manera consistente en todos los navegadores.
- **Es completamente Responsivo:** Bootstrap parte de la filosofía mobile first, todos los componentes fueron diseñados para desplegarse correctamente en un dispositivo móvil y es posible adaptarlos con muy poco esfuerzo a múltiples pantallas y resoluciones.
- **Componentes predefinidos:** Tendrás una serie de elementos predefinidos como botones, listas y contenedores para acelerar tu trabajo evitando la repetición de tareas.

Usando Bootstrap para hacer sitios responsivos

Posiblemente uno de los elementos más relevantes de Bootstrap es su capacidad para adaptarse a múltiples pantallas.

La arquitectura responsiva de Bootstrap nos ofrece varios elementos para construir interfaces, vamos a examinar algunos de ellos para comenzar a crear una aplicación básica.

Contenedores

Los contenedores engloban contenido y lo mantienen agrupado en diferentes tamaños de pantalla, para activar un contenedor solo es necesario agregarle la clase “container” a cualquier elemento:

```
1 <div class="container"> </div>
```

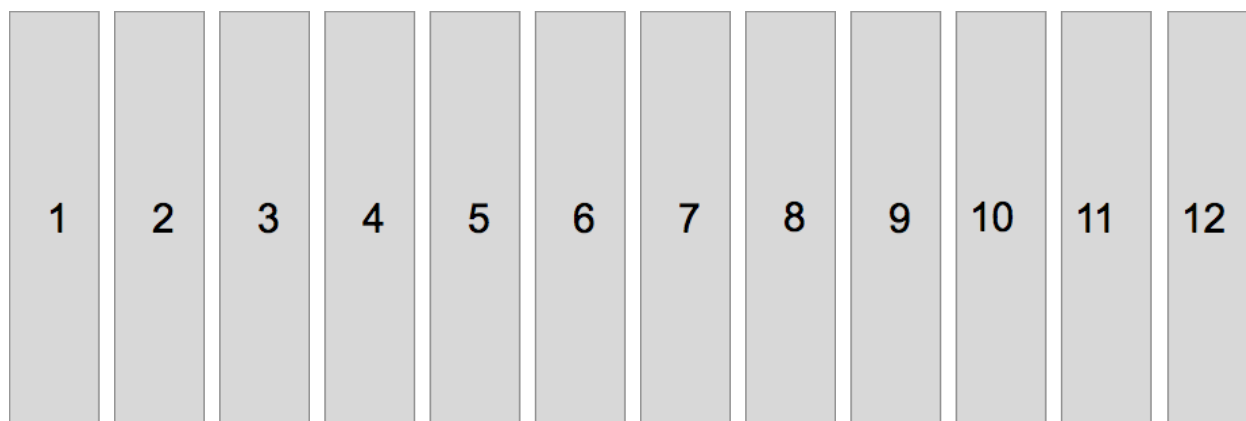
El contenedor cambiará de tamaño según las diferentes resoluciones de pantalla, por ejemplo, en equipos de tamaño reducido no tendrá un tamaño asignado y se ajustará al de la pantalla, por otro lado, en tablets medirá 750px, en equipos de escritorio 970px y en pantallas grandes 1170px, en todos los casos alineado al medio de la pantalla.

El contenedor es perfecto para incluir el contenido de una app o los elementos que necesiten alineación con este contenido tal como menús o barras de navegación.

Matriz de columnas

La base central del sistema responsivo de Bootstrap son las columnas. El concepto de columna es una unidad relativa que se ajusta según el tamaño de la pantalla.

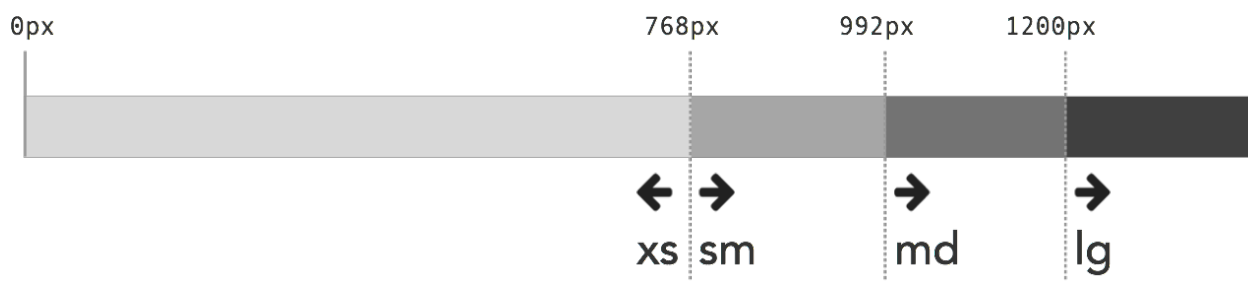
En Bootstrap, una pantalla, independientemente de su tamaño o resolución, se divide de manera horizontal en 12 diferentes columnas, el tamaño específico de la columna es variable según las condiciones del navegador.



Para asignar el ancho de un elemento, se utilizan las columnas como base; al asignar los tamaños en esta medida, te aseguras que van a mantener la misma apariencia en diferentes resoluciones. Por ejemplo, para que un elemento mida la mitad del tamaño de la pantalla se le asignan 6 columnas, así, aunque la app se muestre en una pantalla móvil de 320px o en una pantalla de escritorio de 1024px el elemento se desplegara con la misma proporción en ambos casos.

Soporte para diferentes resoluciones

Además de las columnas, Bootstrap también trabaja con un patrón de 4 diferentes resoluciones que cubren la mayoría de pantallas.



Como se representa en el gráfico cada resolución abarca diferentes posibles condiciones de pantalla desde pantallas reducidas de smartphones (xs) hasta pantallas de alta resolución.

Lo mejor de todo es que dependiendo de la resolución ¡Puedes cambiar la apariencia y posición de los elementos en pantalla! Esto hace tu aplicación realmente responsiva al adaptar el diseño a las condiciones en que se ejecute.

Las clases

para asignar los elementos que acabamos de estudiar, Bootstrap utiliza una nomenclatura que combina diferentes factores y combinaciones, usando este patrón:

1 COL-RESOLUCION-TAMANO

Así por ejemplo, para crear un `<div>` que se despliegue a la mitad de la pantalla en un smartphone, se debería usar el prefijo “col-” para definir una clase relacionada con el ancho, luego “xs-” para relacionarlo con una resolución específica y finalmente el número de columnas que medirá el elemento, ya que la pantalla completa mide 12 columnas, la mitad de la pantalla serían 6, así que el resultado sería:

```
1 <div class="col-xs-6"> </div>
```

Si por otra parte quieres mostrar un elemento que en una pantalla de gran tamaño ocupe todo el ancho disponible, debes usar el prefijo “col-” porque seguimos definiendo columnas, “lg-” porque deseas aplicarlo al mayor tamaño de pantalla y el número 12 porque es el ancho máximo de un elemento Bootstrap.

```
1 <div class="col-lg-12"> </div>
```

Finalmente, Bootstrap también te deja combinar medidas, por ejemplo si quieres que el mismo elemento se vea con ancho de 6 columnas cuando lo usas desde un smartphone pero a 12 columnas cuando lo haces desde una pantalla grande, puedes usar ambas clases, el sistema se encarga de ejecutar cada instrucción en el contexto adecuado

```
1 <div class="col-lg-12 col-xs-6"> </div>
```

Creando una plantilla con Bootstrap

¡Vamos a poner manos a la obra! Crearemos ahora una aplicación AngularJS integrada con Bootstrap.

En este caso, arrancaremos creando una plantilla estática en Bootstrap a la cual le añadiremos luego los elementos interactivos de AngularJS.

Comenzaremos con un código base, en este caso un HTML5 básico donde iremos incluyendo las partes de la aplicación:


```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="utf-8">
6      <title>AngularJS y Bootstrap </title>
7      <meta name="viewport" content="width=device-width">
8  </head>
9
10 <body>
11
12 </body>
13
14 </html>
```

Ya tenemos la base de la aplicación, guarda este documento como “preguntas.html” y ábrelo en tu navegador, aun esta en blanco ¡Pero estamos a punto de cambiar eso!

Antes de comenzar a agregar elementos gráficos, comencemos instalando Bootstrap.

Para instalar este framework necesitamos agregar 2 líneas de código: la primera, incluirá el CSS que modificara la parte visual de nuestro contenido; la segunda, es un archivo JavaScript que contiene herramientas para crear componentes e interactividad.

```
1  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css\
2  /bootstrap.min.css">
3  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js/bootstrap.min.js\
4  "></script>
```

En este caso estamos utilizando la versión 3.2.2 de Bootstrap, te invito a revisar el sitio oficial del proyecto para que encuentres instrucciones detalladas de instalación así como la información con las últimas actualizaciones.

<http://getbootstrap.com/getting-started/>

Agregando contenidos

Tenemos listo un documento con todos los elementos de Bootstrap disponibles y es momento de comenzar a agregar contenidos, comenzaremos incluyendo un <div> con las clases “container” y “text-center” para agrupar el contenido y alinear el texto al medio respectivamente.

```
1 <div class="container text-center">
2
3 </div>
```

También vamos a incluir un encabezado, en este caso incluiremos otro <div> pero esta vez con la clase “jumbotron”, que genera un área destacada para incluir los elementos principales del documento, aprovecharemos esta área para incluir un encabezado.

```
1 <div class="jumbotron">
2   <h1>AngularJS + Bootstrap</h1>
3 </div>
```

Nuestro documento comienza a tomar forma ¡Ya tenemos el encabezado!

AngularJS + Bootstrap

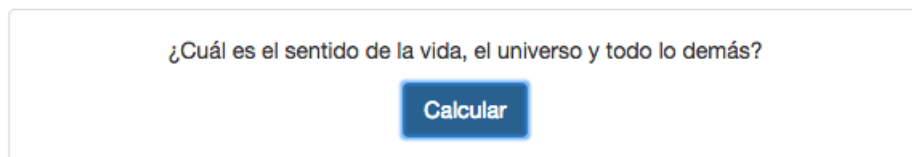
No está nada mal ¿Qué tal si le agregamos algo de contenido a esta app? Usaremos ahora un Panel, un componente de Bootstrap que genera un área que agrupa contenidos.

Utilizaremos las clases “panel” y “panel-default” para definir el componente, también la clase “col-sm-6” para darle un tamaño de 6 columnas o la mitad del área disponible en pantalla; finalmente, incluimos la clase “col-sm-offset-3” para dejar un margen de 3 columnas y alinear el panel al medio.

```
1 <div class="panel panel-default col-sm-6 col-sm-offset-3">
2   <div class="panel-body ">
3     <p>¿Cuál es el sentido de la vida, el universo y todo lo demás?</p>
4
5     <button class="btn btn-primary">Calcular</button>
6   </div>
7 </div>
```

Dentro del panel, se incluye un nuevo `<div>` con la clase “panel-body” que almacena lo que será el contenido del panel. Allí estará un párrafo y un botón, éste último tiene las clases “btn” y “btn-primary” para definirlo como botón y asignarle sus propiedades gráficas.

Tenemos lista nuestra aplicación al menos a nivel de contenido, si la abres en tu navegador debería verse así



Agregar Interactividad en Bootstrap

Bootstrap es mucho más que solo elementos gráficos, también soporte para elementos interactivos, que en muchos casos podemos activar sin necesidad de programar.

Para activar los elementos interactivos, Bootstrap usa la librería jQuery, así que debemos incluir esta línea para agregarla en nuestra aplicación:

```
1 <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js">
2 </script>
```

Asegúrate que esta línea se encuentre dentro de la etiqueta `<head>` y ANTES de la que incluye el JavaScript de Bootstrap, de esa forma te aseguras que jQuery estará disponible desde el primer momento en que se ejecuta la aplicación, si lo colocas en otra posición, es posible que tengas errores de ejecución.

Ahora que jQuery está instalado podemos incluir elementos interactivos, para este ejemplo utilizaremos una ventana modal, un componente que despliega un mensaje destacado dentro de la aplicación.

Agrega el siguiente código en el documento

```
1 <div class="modal fade ventanaModal">
2     <div class="modal-dialog modal-sm">
3         <div class="modal-content">
4             <h1>¿?</h1>
5         </div>
6     </div>
7 </div>
```

Después de incluir este código, notarás que no hay ningún cambio, esto se debe a que el componente que acabas de agregar solo es visible al activarlo. Para que nuestra ventana modal se despliegue vamos a modificar el botón que se encuentra actualmente en la aplicación, asegúrate de reemplazar el código del botón actual por este otro:

```
1 <button class="btn btn-primary" data-toggle="modal" data-target=".ventanaModal">\
2 Calcular</button>
```

Hemos agregado únicamente algunas clases y ninguna programación, pero Bootstrap se encarga del trabajo difícil activando las conductas interactivas automáticamente, abre tu documento en el navegador y al hacer click en el botón “calcular” podrás ver un mensaje similar a este



¡Estupendo! Ya tienes un documento Bootstrap con interactividad y solo tienes un problema restante: la app aun no muestra una respuesta para la pregunta. Esto es un trabajo para...

¡AngularJS!

Instalar AngularJS en Bootstrap

Para integrar AngularJS a este documento, necesitamos primero agregar una línea de código dentro de <head>

```
1 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.12/angular.min.\n2 js"></script>
```

El siguiente paso, es incluir la directiva “ng-app” dentro del documento, en este caso la pondremos en la etiqueta <body> y llamaremos la aplicación “preguntas”

```
1 <body ng-app="preguntas">
```

También, para tener acceso a los elementos de la aplicación incluiremos la directiva “ng-controller” dentro del contenedor general de la app, en este caso lo relacionaremos con el controlador “preguntasController”

```
1 <div class="container text-center" ng-controller="preguntasController">
```

El último paso para inicializar la aplicación es declarar el módulo que la compone:

```
1 <script>\n2   var miApp = angular.module('preguntas', []);\n3 </script>
```

Agregar controladores

Una vez inicializada la aplicación, necesitamos agregar el controlador “preguntasController” que ya se encuentra relacionado desde el HTML

```
1 miApp.controller('preguntasController', ['$scope', function($scope){\n2\n3   $scope.responder = function(){\n4     $scope.respuestaDefinitiva = 6 * 9 - 12\n5   }\n6\n7 }]);
```

Como puedes notar, además de incluir el controlador, he incluido un método, vamos a usarlo para desplegar valores en la ventana modal.

Es necesario hacer dos sencillos cambios en el HTML, el primero es agregar la directiva “ng-click” con el valor “responder()” en el botón de la aplicación para que se ejecute el calculo al momento de mostrar el mensaje. El código final del botón debería ser similar a este:

```
1 <button class="btn btn-primary" data-toggle="modal" data-target=".ventanaModal" \
2 ng-click="responder()">Calcular</button>
```

Solo resta incluir una expresión para mostrar el resultado del calculo de nuestra aplicación, en la ventana modal, busca la etiqueta <h1> y reemplaza el texto “¿?” por una expresión que muestre los valores de “respuestaDefinitiva”

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

Eso es todo, ¡ya hemos terminado la aplicación! Guarda tu documento y al abrirlo en el navegador veras el resultado final con la respuesta que buscábamos al presionar el botón



Tenemos ahora una aplicación totalmente funcional y en caso que te queden dudas sobre la validez del resultado, te dejo un enlace con más información sobre la respuesta para que impresiones a tus amigos con datos frikis ;)

http://es.wikipedia.org/wiki/El_sentido_de_la_vida,_el_universo_y_todo_lo_dem%C3%A1s

Administrar plantillas y rutas

Posiblemente una de las herramientas más poderosas de AngularJS es su capacidad de cargar plantillas y administrar las rutas del navegador. Esta técnica, comúnmente llamada “routing” nos permite ejecutar aplicaciones completas con solo un documento HTML.

Las aplicaciones web tradicionales cargan un documento nuevo por cada pantalla que muestran y en algunos casos, se recargan únicamente para realizar procesos simples. Cada vez que pasas de una página a otra , así sea que solo necesites mostrar un cambio simple, volverás a descargar todos los archivos CSS, imágenes, JavaScript y elementos de HTML comunes. Este proceso es ineficiente porque el usuario termina descargando los mismos datos una y otra vez

Al utilizar plantillas de AngularJS, solo descargaremos nuevos contenidos, conservando disponibles todas las imágenes, código y datos de la aplicación. Debido a que no duplicas descargas, el ancho de banda baja dramáticamente, además, al cargar solamente la información necesaria, los archivos son más pequeños, lo que disminuye también los tiempos de respuesta.

Indexación de AngularJS en motores de búsqueda

Tradicionalmente este tipo de aplicaciones usan llamadas asincrónicas AJAX para la carga de sus datos, AngularJS utiliza también esta tecnología pero va más allá, utilizando las nuevas API's de HTML para modificar la ruta que aparece en el navegador , puede, además de cargar datos con el máximo de eficiencia, generar aplicaciones que son amigables con la indexación en motores de búsqueda

Un mito común sobre las aplicaciones basadas en AngularJS es que al utilizar las plantillas con carga asincrónica los motores de búsqueda no te indexarán correctamente y prácticamente serás invisible en la red. Esto es totalmente falso ya que desde mediados del 2014 los robots de google leen, procesan e indexan sin ningún problema este tipo de aplicaciones y todos sus enlaces. Puedes profundizar más sobre este tema en el Blog oficial de desarrolladores de Google (en inglés)

<http://googlewebmastercentral.blogspot.com/2014/05/understanding-web-pages-better.html>

Dejemos la teoría a un lado por un momento, ¡Vamos a trabajar! Ya verás que cuando domines los contenidos de este capítulo estarás preparado para comenzar a hacer ¡Verdaderas aplicaciones web!

Comenzaremos de inmediato a crear el ejercicio para este capítulo, vamos a crear un archivo con el nombre “routing.html” y le incluiremos este código:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Routing</title>
6      <script src="angular.min.js"></script>
7      <style>
8          ul{margin: 0; padding: 0; margin: auto; width: 400px;}
9          li{float: left; height: 30px; list-style-type: none; width: 100px;}
10         a{display: block}
11         h2{color:red}
12         div{clear: both}
13     </style>
14 </head>
15
16 <body ng-app="routingApp">
17
18     <div id="menu"></div>
19     <div id="contenedor"></div>
20
21     <script>
22         var miRouting = angular.module('routingApp', []);
23     </script>
24
25 </body>
26
27 </html>
```

En este ejemplo ya tenemos una aplicación con el nombre “routingApp” correctamente instalada e inicializada. Tenemos también dos <div> con los identificadores “menu” y “contenedor”, usaremos más adelante estos elementos para insertar contenidos, pero por ahora, solo son contenedores vacíos.

Instalando ngRoute

El código original de AngularJS incluye una serie de operaciones comunes para la mayoría de los escenarios posibles, pero para elementos más avanzados como el manejo de rutas y plantillas se requiere que instalemos un módulo adicional con rutinas especializadas.

La funcionalidad adicional que vamos a instalar se encuentra en el repositorio de código oficial de AngularJS, puedes encontrarla en el sitio oficial del proyecto, buscando la opción “Browse additional modules”

Download AngularJS


Branch ?

Build ?

CDN ?

Bower ?

npm

Extra: [Browse additional modules](#) 

[Previous Versions](#)

Al abrir el enlace veras una lista de archivos, en este caso usaremos el que tiene el nombre “angular-route.min.js”

angular-animate.min.js	17-Mar-
angular-animate.min.js.map	17-Mar-
angular-aria.js	17-Mar-
angular-aria.min.js	17-Mar-
angular-aria.min.js.map	17-Mar-
angular-cookies.js	17-Mar-
angular-cookies.min.js	17-Mar-
angular-cookies.min.js.map	17-Mar-
angular-csp.css	17-Mar-
angular-loader.js	17-Mar-
angular-loader.min.js	17-Mar-
angular-loader.min.js.map	17-Mar-
angular-messages.js	17-Mar-
angular-messages.min.js	17-Mar-
angular-messages.min.js.map	17-Mar-
angular-mocks.js	17-Mar-
angular-resource.js	17-Mar-
angular-resource.min.js	17-Mar-
angular-resource.min.js.map	17-Mar-
angular-route.js	17-Mar-
angular-route.min.js	17-Mar-
angular-route.min.js.map	17-Mar-
angular-sanitize.js	17-Mar-
angular-sanitize.min.js	17-Mar-
angular-sanitize.min.js.map	17-Mar-
angular-scenario.js	17-Mar-
angular-touch.js	17-Mar-
angular-touch.min.js	17-Mar-
angular-touch.min.js.map	17-Mar-

Puedes descargar el archivo o simplemente copiar su dirección al servidor de google, en ambos casos, debes agregar al documento de tu aplicación una etiqueta <script> indicando

la ruta, en este caso usaremos un enlace externo

```
1 <script src="https://code.angularjs.org/1.3.14/angular-route.min.js"></script>
```

Recuerda adaptar la dirección que se usa en este ejemplo para reflejar la posición de tu archivo y/o la versión de AngularJS que uses.

El módulo “angular-route” nos permite administrar rutas y cargar plantillas dinámicamente, para esto, utiliza un servicio “ngRoute” que después de incluir el código anterior estará disponible en nuestro documento.

Para que nuestra aplicación tenga acceso a dicho servicio debemos inyectarlo en el momento de declarar el módulo, esto lo haremos reemplazando el arreglo vacío al final del comando “[]” por “[ngRoute]”

```
1 var miRouting = angular.module('routingApp', ['ngRoute']);
```

Una vez realizadas estas dos modificaciones al código, estarás listo para administrar rutas y realizar todos los ejercicios de este capítulo.

Toma en cuenta que vamos a cargar archivos, así que necesitaremos hacerlo usando un servidor local. Si tienes alguna duda sobre este tema, recuerda que en el capítulo sobre acceso a datos externos puedes encontrar instrucciones detalladas al respecto.

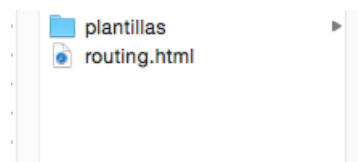
Preparando las plantillas

El concepto de plantillas en AngularJS se refiere al uso de pequeños archivos HTML que se cargan en una área específica y sirven para desplegar contenidos que cambian a lo largo de la navegación de la aplicación.

Las plantillas son modulares y encapsuladas, así que pueden entrar y salir en cualquier momento, dependiendo de las condiciones de navegación, más adelante examinaremos esta conducta en detalle.

Por ahora, vamos a crear los archivos que usaremos como plantillas y los organizaremos de manera que sea sencillo de cargar en la aplicación, en este ejercicio vamos a usar 4 diferentes documentos que serán usados como plantillas.

Comenzaremos creando una carpeta que llamaremos “plantillas” junto al archivo “routing.html” que acabamos de generar, aquí almacenaremos todos los archivos que vamos a crear.



El primer archivo que vamos a crear dentro de la carpeta “plantillas” se llamará “menu.html” y tendrá este contenido:

```
1 <ul>
2     <li> <a href="#">Inicio</a></li>
3     <li> <a href="#/seccion1">Sección 1</a></li>
4     <li> <a href="#/seccion2">Sección 2</a></li>
5 </ul>
```

Usaremos más adelante este código para generar un menú de navegación. Ahora crearemos un archivo llamado “inicio.html” que tendrá los contenidos que mostraremos por defecto en pantalla y usaras dentro de él únicamente esta línea de código:

```
1 <h1>Tienda de pantalones</h1>
```

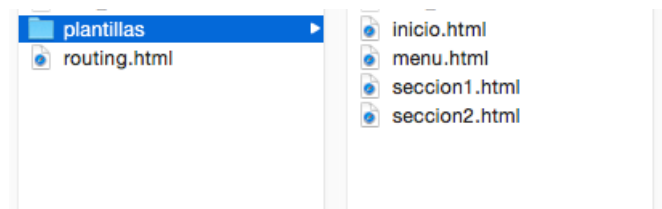
El siguiente archivo se llamara ‘seccion1.html’ y debes agregarle este texto:

```
1 <h1>SECCIÓN 1</h1>
2 <p>Estás viendo la primera sección</p>
```

El archivo restante se llamara “seccion2.html” y será casi igual al anterior, solo modificaremos ligeramente su contenido y nombre para que refleje una nueva sección:

```
1 <h1>SECCIÓN 2</h1>
2 <p>Estás viendo la segunda sección</p>
```

Al final los archivos de tu proyecto deberían tener esta estructura



Usando includes

En AngularJS tenemos diferentes tipos de plantillas y los includes son una de ellas. Su funcionamiento es sencillo: inyectan un trozo de código dentro de un área de la aplicación.

Los includes son la forma más simple de plantilla y se usan para insertar elementos que no van a cambiar durante la navegación, por ejemplo menús, encabezados o pies de página.

Para activar un include usaremos la directiva “ng-include” que permite cargar los contenidos HTML de un archivo externo dentro de la etiqueta en que se encuentre. A manera de estructura general, se utiliza esta sintaxis:

```
1 <ETIQUETA ng-include=" 'RUTA_AL_DOCUMENTO.HTML' " > </ETIQUETA>
```

Al usar este código, “ng-include” tomará todos los contenidos del archivo “documento.html” y los incluirá íntegramente dentro de la etiqueta que le corresponde.

Debido a que las plantillas se insertan dentro de un documento existente, no es necesario que agregues en ellas elementos estructurales del HTML como el doctype, <header> o <body>, solamente incluyes lo que necesitas mostrar, de esta forma ahorras ancho de banda.

Toma en cuenta un detalle importante de la sintaxis de esta directiva, si revisas el código que incluí más adelante notarás que para asignar los valores de la ruta, además de las comillas dobles ("") que se usan de manera regular en HTML5, he incluido unas comillas simples (') , este detalle es un requisito obligatorio para que funcione correctamente.

La directiva “ng-include” leerá todos los valores dentro de las comillas dobles ("") en formato JavaScript y su correspondiente sintaxis, la ruta que incluimos en este caso es una cadena de texto, por eso las comillas simples ('). Si deseas también puedes incluir variables y operaciones complejas siempre y cuando cumplan con la sintaxis correcta de JavaScript.

Aplicando estos conceptos a nuestro ejercicio, vamos a incluir en la aplicación el archivo “menu.html” que se encuentra en la carpeta “plantillas”. Usaremos entonces, la directiva “ng-include” y asignaremos la cadena de texto que define la ruta a este documento dentro de comillas simples

```
1 <div id="menu" ng-include=" 'plantillas/menu.html' "></div>
```

Ahora, al ejecutar el archivo “routing.html” se mostrara también el código correspondiente al menú, como una parte integral de la aplicación.

[Inicio](#)[Sección 1](#)[Sección 2](#)

Tienda de pantalones

Nuestra app ya esta usando plantillas a la perfección a través de includes, vamos a ver una forma más sofisticada de plantilla con las vistas de AngularJS.

Vistas

Las vistas funcionan muy parecido a los includes pero se diferencian en un punto fundamental: el archivo que cargan depende totalmente de la ruta y configuración de la aplicación, o

sea el elemento que muestran es totalmente dinámico y puede cambiar según la navegación del usuario.

Utilizando vistas podemos cargar un número ilimitado de contenidos a lo largo del uso de la aplicación. Este componente se encuentra optimizado para eliminar cualquier consumo de memoria de la vista anterior antes de cargar la siguiente, así que tu aplicación permanece siempre eficiente y sin procesos innecesarios.

Para activar las vistas solamente debes definir la directiva “ng-view” dentro de una etiqueta. En este ejercicio la usaremos en el <div> con la identificación “controlador”

```
1 <div id="contenedor" ng-view></div>
```

Toma en cuenta que solo debe existir una vista por aplicación y que no puedes anidar vistas dentro de otras, pero ya veras que una sola vista es lo suficientemente flexible para cumplir con todas las necesidades de tu app.

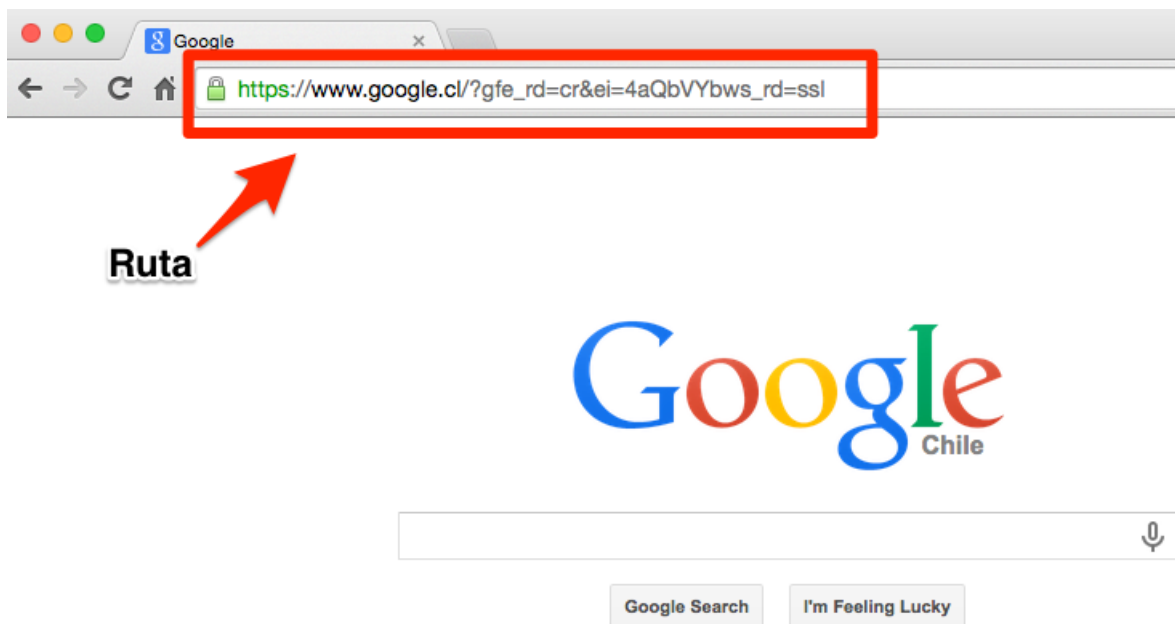
La vista en si misma no tiene control sobre los elementos que despliega, por eso solo se asigna la directiva, sin ningún valor.

Piensa en este componente como una pantalla de TV que puede mostrar muchos contenidos distintos en un solo lugar.

El contenido que muestra una vista, realmente se administra desde la configuración de rutas y es justo ese tema del que hablaremos a continuación.

Rutas

AngularJS permite administrar las rutas o URLs en el que se muestra nuestra aplicación. Generalmente puedes encontrar la ruta de un documento en la parte superior de tu navegador, en la llamada “barra de direcciones”



AngularJS permite crear aplicaciones en un solo documento con todas las ventajas que eso conlleva, sin embargo, al usar un solo documento, técnicamente solo tendrías una ruta para todos los contenidos de tu aplicación.

La configuración de rutas de AngularJS resuelve ese problema y nos permite generar diferentes direcciones para diferentes contenidos, así los motores de búsqueda podrán indexar todas las partes de tu aplicación.

Definiendo las rutas

El primer paso para administrar las rutas es configurar la aplicación haciendo un mapeo de las posibles rutas y a partir de ahí, definir las conductas que tendrá según el contenido solicitado.

Usaremos la instancia donde creamos la aplicación y a partir de allí definiremos el método “config”. En nuestro ejercicio la instancia se llama “miRouting” así que usaremos este código

```
1 miRouting.config(['$routeProvider', function(){  
2  
3 }]);
```

La sintaxis del método “config()” es usar como argumento un arreglo con 2 valores: el primero es “\$routeProvider”, un objeto especial de AngularJS que permite interactuar con las rutas de la aplicación y que está accesible gracias a la librería adicional que instalamos al inicio del capítulo, el segundo valor es una función, allí dentro incluiremos todas las propiedades de configuración.

Dentro de la función utilizaremos el objeto “\$routeProvider” para gestionar las rutas. Tendremos dos diferentes comandos para aplicar en dicho objeto, vamos a revisarlos individualmente

when

Literalmente significa “cuando” de manera condicional, aplicado al sistema de rutas esta propiedad nos permite generar una conducta que solo se ejecute cuando nos encontramos en una ruta en particular.

De manera general la sintaxis de este comando funciona así:

```
1 $routeProvider.when('RUTA', { PROPIEDADES DE LA PLANTILLA }).
```

Gracias al objeto “\$routeProvider” tenemos acceso a la ruta actual y usaremos “when” a manera de operador condicional. El parámetro “RUTA” es una cadena de texto con la ruta que vamos a evaluar, si nuestra aplicación se encuentra allí, se activará una plantilla, de ella se encarga la última parte de este comando, “{PROPIEDADES DE LA PLANTILLA }” un objeto que contiene valores de configuración.

En este último objeto podemos incluir diferentes valores, los más importantes son: - **templateURL**: una cadena de texto que define la ruta a la plantilla que deseas mostrar. - **controller**: permite asignar un controlador dedicado a esta plantilla.

Veamos como funcionarían estos conceptos en nuestro ejercicio, por ejemplo, si queremos que plantilla “seccion1.html” se cargue en la vista de la aplicación (“ng-view”), cada vez que el usuario acceda a la ruta “/seccion1”, usaremos estos valores:

```
1 $routeProvider.when('/seccion1', {  
2     templateUrl: 'plantillas/seccion1.html',  
3 })
```

Para incluir nuevas rutas, no es necesario volver a definir otro bloque de comandos, únicamente encadenas otra condición “when” al final y repites el proceso. Sin duda tus aplicaciones van a necesitar más de una ruta y con esta técnica puedes agregar todas las que quieras.

Aplicado a nuestro ejercicio, para agregar una nueva ruta, esta vez mostrando la plantilla “seccion2.html” al entrar a la ruta “/seccion2”, modificaremos el código anterior por este

```
1 $routeProvider.when('/seccion1', {  
2     templateUrl: 'plantillas/seccion1.html',  
3 })  
4 .when('/seccion2', {  
5     templateUrl: 'plantillas/seccion2.html',  
6 })
```

De seguro que ya lo notaste, pero si no, pon atención al punto que usamos justo antes de “when”, esa es la forma que podemos agregar rutas ilimitadamente, asegúrate de siempre incluirlo, de lo contrario tendrás problemas para ejecutar el código.

otherwise

Supongamos que tu aplicación ya tiene definidas todas las posibles rutas ¿que pasa si un usuario escribe por error una ruta con un nombre incorrecto? En este momento, al no estar definida ninguna instrucción para ese tipo de casos, tu aplicación no mostrará nada o desplegará contenido incompleto.

¡No podemos permitir que algo tan trivial nos haga quedar mal!

Para estos casos especiales tenemos la propiedad “otherwise” que es similar a “when”, pero esta se ejecuta únicamente si la ruta actual NO CONCUERDA con ninguna de las definidas.

La sintaxis general de “otherwise” sería así

```
1 $routeProvider.otherwise({ PROPIEDADES DE LA PLANTILLA })
```

Debido a que esta propiedad actúa a manera de seguro contra errores, no necesita que le definamos ninguna ruta para evaluar, además usa una propiedad especial “redirectTo” para direccionar el tráfico:

```
1 redirectTo: '/',
```

En este caso, la propiedad está configurada para enviar todo el tráfico a la raíz de la aplicación.

Ahora que conoces los comandos para administrar rutas, y unificando todos los diferentes conceptos que hemos estudiado hasta ahora, la configuración para nuestro ejercicio actual quedaría así


```
1 miRouting.config(['$routeProvider',
2     function ($routeProvider) { $routeProvider.
3         when('/seccion1', {
4             templateUrl: 'plantillas/seccion1.html',
5         })
6         .when('/seccion2', {
7             templateUrl: 'plantillas/seccion2.html',
8         })
9         .otherwise({
10             redirectTo: '/',
11             templateUrl: 'plantillas/inicio.html',
12         });
13 }]);
```

En este caso tenemos 2 diferentes rutas habilitadas para mostrar las secciones del ejercicio, en caso de que el usuario entre a una ruta diferente lo redireccionaremos a la raíz de la aplicación y mostraremos la plantilla base “inicio.html”.

Lo sé, ha sido una larga explicación, pero valió la pena ¡La aplicación ya está funcional y usando rutas!

Abre el documento “routing.html” en el navegador, las rutas que definimos vienen listas en el menú y podrás ver como ya está funcionando cada una de ella y además, si entras a una dirección que no esta definida, el sistema te redireccionará a la pantalla inicial.

Tenemos una aplicación que ya puede trabajar sin problema en la mayoría de los casos básicos, mostrando secciones y contenidos dinámicamente... Pero sabemos que un sitio básico no es suficiente para ti ¿no es cierto?

Que tal si le agregamos controladores a estas plantillas para que tengan capacidad de realizar cálculos y operaciones avanzadas

Cargando controladores

Ya aprendimos a cargar plantillas dinámicamente, pero AngularJS puede hacer mucho más que eso, también podemos asignar diferentes controladores a cada una de las plantillas que cargamos.

El proceso es sencillo, pero se divide en 3 pasos que debes repetir todas las veces:

1. Crear un archivo para el controlador
2. Incluir el archivo del controlador dentro de la app
3. Relacionar el controlador con una plantilla/ruta

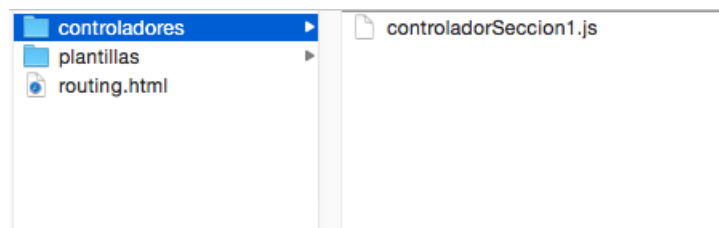
Revisemos estos tres pasos por separado

Crear el archivo del controlador

Para mantener una aplicación ordenada y modular, lo mejor que puedes hacer es separar cada controlador en un archivo diferente, te será más fácil administrar tu código y hacer cambios rápidos.

Puedes crear este archivo donde quieras, pero te recomiendo hacerlo dentro en una carpeta llamada por ejemplo “controladores” así tendrás un lugar para guardar archivos de este tipo.

En este ejercicio vamos a crear un controlador para la primera sección de la app, así que dentro de la carpeta que acabas de hacer, crea un nuevo archivo llamado “controladorSeccion1.js”



Dentro de este archivo, crearemos un controlador simple llamado ‘miControlador’ y dentro de él almacenaremos una variable llamada ‘titulo’

```
1 miRouting.controller('miControlador', function($scope){  
2  
3     $scope.mensaje = 'Esta es una variable almacenada en el controlador';  
4  
5 })
```

Incluir el controlador dentro de la app

Después de crear el controlador, tenemos que hacer que este disponible dentro de la aplicación, lo único que debes hacer es agregar una etiqueta <script> con la ruta correspondiente.

```
1 <script src="controladores/controladorSeccion1.js"></script>
```

Toma en cuenta que esta línea debes incluirla únicamente en el archivo que ejecuta la aplicación, en este ejercicio sería el archivo “routing.html”.

Es un error muy común olvidar este simple proceso y esto genera errores importantes, si al usar por primera vez un controlador tienes problemas, asegúrate que realizaste este paso correctamente.

Relacionar el controlador con una ruta

El ultimo paso es el más sencillo, solo debes buscar el bloque de configuración de las rutas, agregar la propiedad “controller” y asignarle como valor el nombre del controlador.

En nuestro caso debemos modificar el código de esta forma

```
1 when('/seccion1', {
2     templateUrl: 'plantillas/seccion1.html',
3     controller: 'miControlador'
4 })
```

Pon atención que a diferencia de la propiedad ‘templateURL’ que usa el nombre de un archivo, para la propiedad ‘controller’ solo estamos usando el nombre del controlador.

Ahora al entrar a la ruta “/seccion1” además de desplegar la plantilla “seccion1.html” también activaremos automáticamente el controlador “miControlador”.

Vamos a probar si todo funciona correctamente, abre el archivo “seccion1.html” y agrega esta expresión:

```
1 <h2>{{mensaje}}</h2>
```

Guarda todos los archivos que has modificado y abre tu aplicación en el navegador, al hacer click en el apartado “Sección 1” tendrás funcionando el nuevo controlador y desde allí se carga el texto que despliega la expresión que acabas de agregar.

[Inicio](#) [Sección 1](#) [Sección 2](#)

SECCIÓN 1

Estás viendo la primera sección

Esta es una variable almacenada en el controlador

Quiero hacer una pausa aquí ¿notaste que el mensaje esta en rojo? Eso es porque el CSS de tu documento se aplica en todos los elementos de la aplicación AngularJS, incluso en los archivos que cargas como plantilla o en las expresiones variables. ¿Genial no?

Nuestra aplicación ya ha mejorado bastante, controlamos rutas y podemos asignar controladores para cada pantalla, pero aun debes definir cada ruta manualmente ¿como mostrar las rutas para cientos de artículos?

Tranquilo, no tendrás que escribir cientos de rutas, solo vamos a hacerlas más inteligentes

Rutas dinámicas

Las rutas dinámicas permiten generar nuevas rutas basadas en valores cambiantes y utilizar la información del URL para adaptar los contenidos.

Pensemos en este caso: tienes una tienda de pantalones con una lista de cientos de diferentes colores y estilos, cuando el usuario encuentra uno que le gusta, puede ver los detalles de esa prenda en especial.

Crear manualmente las cientos de posibles rutas para una aplicación así, es demasiado complejo, para estos casos tenemos las rutas dinámicas.

Comencemos a implementar este concepto agregando las rutas y usaremos como ejemplo la tienda de pantalones. Abre el archivo “inicio.html” y agrega este texto

```
1 <a href="#/pantalones/rojos">Pantalones Rojos</a>
2 <a href="#/pantalones/azules">Pantalones Azules</a>
3 <a href="#/pantalones/cafes">Pantalones Cafés</a>
```

Como puedes ver tenemos aquí tres diferentes enlaces, cada uno de ellos apunta a una carpeta llamada “pantalones” dentro de la cual tendremos un apartado correspondiente a cada color.

Supongo que te preguntarás si es necesario crear esa carpeta, pues no, ¡todo se administra desde AngularJS! este es el verdadero poder que tenemos al controlar las rutas, puedes crear direcciones fáciles de recordar, con sentido para los usuarios y de paso incluir palabras clave o keywords

¡a google le encantará indexar este tipo de enlaces!

Vamos ahora a administrar estas tres nuevas rutas enviándolas a un mismo documento donde se mostrarán los detalles de cada una. Debes crear un nuevo archivo en la carpeta “plantillas” con el nombre “detalles.html”.

En este documento, por ahora solo vamos a agregar este código

```
1 <h1>Detalles</h1>
2 <p>Estás viendo los detalles de un pantalón</p>
```

Ahora que tenemos la plantilla, el siguiente paso es dirigir allí el tráfico de estos tres enlaces agregando este nuevo código a la configuración de rutas de la aplicación.

```
1 .when('/pantalones/:color', {  
2     templateUrl: 'plantillas/detalles.html',  
3 })
```

Si revisas con cuidado usamos el mismo patrón de las opciones anteriores: tenemos a “when” para evaluar una ruta y estamos definiendo la plantilla con el parámetro “templateURL”.

El cambio en este comando viene al final de la ruta a evaluar, donde usamos el valor “:color”. En la sintaxis de AngularJS para rutas, al usar los dos puntos seguidos de una palabra la conviertes en una variable y creas una ruta dinámica.

El comportamiento de este comando será evaluar el inicio de la ruta e ignorar la parte asignada a la variable, en este caso cualquier ruta que comience con “/pantalones/”, sin importar que otro texto contenga, desplegará la plantilla “detalles.html”

Guarda todos los archivos y abre tu aplicación en el navegador, prueba por ejemplo hacer click en cualquiera de los enlaces, no importa ahora el color de los pantalones siempre tendrás como resultado la página de detalles



¡Excelente! ya estas controlando una ruta dinámica, ahora puedes tener un número ilimitado de combinaciones de rutas dentro de la carpeta pantalones que siempre llegara al mismo documento, no será necesario crear una por cada color.

Solo falta un ultimo detalle para que este ejercicio tenga una funcionalidad completa ¿que tal si leemos la información de la ruta y mostramos contenido diferente en cada una?

Leer variables del URL

Como lo puedes sospechar al crear rutas dinámicas estamos generando variables y esta información esta disponible dentro de la aplicación, lo que significa que no solo podemos hacer rutas adaptables, también podemos saber exactamente que información buscaba el usuario.

En nuestro ejercicio vamos a obtener el color del pantalón que solicitó el usuario.

Este proceso necesita un controlador para procesar los datos, así que para no extender de más este ejercicio vamos a asignar el controlador que creamos anteriormente, solamente debes modificar el código que administra la ruta dinámica incluyendo la propiedad 'controller'

```
1 .when('/pantalones/:color', {  
2     templateUrl: 'plantillas/detalles.html',  
3     controller: 'miControlador'  
4 })
```

Debido a que este controlador ya se encuentra instalado y disponible no es necesario modificar la aplicación, pero si vamos a agregar una nueva variable dentro del controlador para acceder los valores que nos llegan de la ruta. Abre el archivo 'controladorSeccion1.js' y modifica el código actual para que quede así:

```
1 miRouting.controller('miControlador', function($scope, $routeParams){  
2     $scope.titulo = 'Catálogo de pantalones';  
3     $scope.colorActual = $routeParams.color;  
4 })
```

He realizado dos cambios al controlador original: Primero inyecté un objeto llamado "\$routeParams" que contiene la información de la ruta. Segundo, he creado una variable con el nombre 'colorActual' donde uso éste objeto y desde allí obtengo el valor dinámico de la ruta. Recuerda que en este caso he usado la propiedad "color" porque así es como se llama la variable dentro de la ruta dinámica (":color")

Tenemos casi terminado este ejercicio, solo falta desplegar los datos que estamos extrayendo de la ruta, abre el archivo "detalles.html" y modifica el párrafo actual por este que usa una expresión para representar la variable que acabamos de crear en el controlador

```
1 <p>Estás viendo los detalles de los pantalones {{colorActual}}</p>
```

Guarda todos los documentos y prueba tu aplicación, ahora veras un mensaje diferente por cada uno de los colores de pantalón que elijas



Este es el momento en que ya puedes empezar a sentirte un verdadero desarrollador AngularJS, acabas de aplicar una serie de conceptos avanzados, ya verás que cuando trabajes tus aplicaciones en el mundo real utilizarás la mayoría de los temas que viste en este capítulo. Descansa un momento, tomate una cerveza (o tu bebida favorita) ¡Te lo has ganado!

Conclusión y despedida

Hemos llegado al final del libro y espero que para este punto tengas una noción detallada de AngularJS , sus componentes y funcionamiento.

A lo largo de todos los ejemplos y ejercicios del libro, aprendiste a ejecutar procesos con los controladores y eventos, a relacionar valores usando el objeto `$scope`, desplegar datos con expresiones y hasta a generar una aplicación con múltiples vistas en un solo documento a través de las rutas.

Hemos avanzado un largo trecho desde las primeras páginas y lo más importante es que ya tienes un nivel que va mucho más profundo de lo que podrías encontrar en un tutorial de internet.

¡Felicitaciones!

Ya puedes decir que eres un desarrollador AngularJS, tal vez (solo tal vez) esto no te convierta el alma de la fiesta, pero estoy seguro que tus clientes y futuros proyectos lo agradecerán ;)

Próximos pasos

Un verdadero guerrero del código jamás está satisfecho y aunque espero que lo que aprendiste acá te sea de gran utilidad, también espero que no sea suficiente, que te deje con ganas de seguir investigando y esto solo sea el inicio de tu camino para especializarte en AngularJS.

El siguiente paso no te lo puede dar este humilde libro sino el tiempo, entre más aplicaciones en AngularJS desarrolles más trucos aprenderás.

Documentación oficial

AngularJS tiene una documentación detallada sobre todas sus funcionalidades, ahora que conoces bien las bases te será mucho más sencillo de comprender como funcionan los elementos más avanzados.

No olvides visitar con frecuencia la documentación oficial, allí no solo encontrarás toda la información sobre los componentes de AngularJS, también te mantendrás al tanto de los últimos avances y nuevas versiones.

<https://docs.angularjs.org/api>

Si prefieres mantener una versión de toda la documentación almacenada en tu ordenador sin necesidad de conexión a internet puedes usar los programas Dash para MacOS o Zeal para PC/Linux.

<https://kapeli.com/dash>

<http://zealdocs.org>

Angular 2

Al momento de escribir este libro se está trabajando en una segunda versión de AngularJS, la cual por ahora, se encuentra en una versión preliminar exclusivamente para desarrolladores.

La futura versión de este framework se encuentra en desarrollo, lo que significa que pueden ocurrir cambios radicales en su sintaxis, capacidades y propiedades; te recomiendo esperar hasta el lanzamiento de primera versión estable para utilizarlo en tus sitios web de manera segura.

Basado en la información disponible en este momento, las características principales de AngularJS 2 serán: mejoras importantes en la velocidad, nuevos componentes y sintaxis; Todo esto será compatible con el código elaborado en AngularJS 1, lo que te permitirá seguir aplicando lo que has aprendido en este libro en el futuro.

Ya que el proyecto está en desarrollo y puede cambiar en el futuro, te recomiendo que revises la información actualizada en el sitio oficial de AngularJS

<https://angularjs.org/>

Cursos Online

Si te quedaron dudas sobre algún tema, quieres profundizar un poco más sobre alguna técnica o simplemente prefieres una guía personalizada te invito a que visites el sitio oficial del libro para consultar las fechas y horarios para el curso oficial basado en este libro.

manualdelguerrero.com/angularjs/cursos

En el curso en línea además de abarcar todos los temas del libro, exploraremos algunos temas avanzados, casos prácticos y responderé personalmente y en vivo cualquier duda que tengas sobre este libro y AngularJS.

Despedida

Espero sinceramente que tu nivel de conocimientos sobre AngularJS no solo sea mucho mejor del que tenías al inicio, mi objetivo principal era presentarlo de una forma entretenida para que te entusiasmes con esta tecnología y que desees seguir aprendiendo...si llegaste tu lectura hasta la última página ¡Espero haberlo logrado!

Nos vemos en un nuevo libro, si este te gustó y te fue de utilidad te agradecería mucho si dejas un review positivo en Amazon y lo compartes con tus colegas!