

Manual de introducción a



Antonio Javier Gallego Sánchez



Tabla de contenido

Contenidos	1.1
Introducción	1.2
Instalación	1.3
Conceptos básicos	1.4
Nuestro primer proyecto	1.4.1
Estructura de carpetas	1.4.2
Código básico	1.4.3
Componentes	1.5
Área de contenidos	1.5.1
Cabeceras	1.5.2
Pies de página	1.5.3
Botones y enlaces	1.5.4
Listados	1.5.5
Tarjetas	1.5.6
Formularios	1.5.7
Iconos	1.5.8
Arquitectura de una aplicación	1.6
Configuración y rutas	1.6.1
Controladores	1.6.2
Vistas	1.6.3
Servicios	1.6.4
Directivas	1.6.5
Ejemplo completo	1.6.6
Plugins	1.7
Publicación	1.8
Más información	1.9
Ejercicios 1	1.10
Ejercicios 2	1.11

Contenidos

Manual de introducción a Ionic.

Introducción

Ionic es un *framework open source* construido usando HTML5, CSS3 y Javascript para el desarrollo de *aplicaciones híbridas* para dispositivos móviles. La librería está orientada únicamente a su visualización en dispositivos móviles, es decir, no es una librería *responsive* tipo Bootstrap que se adapte a distintos tamaños de pantalla. Todo esto hace que los componentes tengan una apariencia más similar a la nativa, que funcionen más rápido, y por lo tanto que mejore la experiencia del usuario que use la aplicación.



Ionic incluye una completa librería de componentes, estilos y animaciones que simulan el aspecto nativo de las distintas plataformas. Estos componentes adoptarán automáticamente la apariencia nativa del dispositivo en el que se visualicen. Por ejemplo, si incluimos un *checkbox* en un formulario, al compilar y visualizar dicha aplicación en Android, el *checkbox* adoptará un aspecto distinto al que mostraría en iOS.

Otra característica interesante de Ionic es que utiliza Angular para el desarrollo del código dinámico de la aplicación. Ionic nos permite crear una aplicación sin necesidad de escribir ni una línea de Javascript ni de Angular, solamente con CSS y HTML. Pero para aplicaciones más complejas podemos usar Angular, lo que nos permitirá una mayor potencia a la hora de crear contenidos dinámicos, para por ejemplo mostrar un listado a partir de datos cargados de Internet o almacenados en el móvil. Además Angular nos permite estructurar el código de la aplicación y modularizar las distintas partes del mismo siguiendo patrones de desarrollo como MVC, donde tendríamos el código separado en modelos de datos, vistas y controladores.



Las *aplicaciones híbridas* son básicamente páginas web que se ejecutan o renderizan dentro de un navegador web de una aplicación nativa. Es decir, el desarrollador crea la aplicación usando código web (como HTML, CSS, Javascript, etc.) y el *framework* Ionic en este caso. Una vez finalizado el desarrollo, este código se copiará dentro de una aplicación nativa, que consistirá en una única pantalla de tipo visor web en la que se cargará el código de la página web. Esta técnica nos permite generar a partir de código web una aplicación compilada como si fuera nativa, que se puede instalar en cualquier dispositivo y que la podemos publicar en el *market* de aplicaciones.

Además, las *aplicaciones híbridas* tienen muchas otras ventajas, pero la principal es la velocidad de desarrollo, ya que con un único código web podemos generar aplicaciones para todas la plataformas: Android, iOS, Windows Phone, etc. Simplemente tendremos que crear la aplicación nativa de cada plataforma con un visor web e incluir nuestro código dentro.

Ionic, gracias a que viene integrado con Cordova (o PhoneGap), también permite el acceso a las características nativas de los dispositivos. Es decir, desde el código web podremos hacer uso de los sensores del dispositivo como GPS, cámara, acelerómetros, brújula, etc. o incluso a las opciones de almacenamiento o la agenda de contactos.



Ahora que ya hemos visto que es Ionic y lo que puedes hacer con él, vamos a empezar explicando como instalar esta herramienta y como crear nuestra primera aplicación.

Instalación

En esta sección vamos a ver en detalle todo el proceso de instalación de Ionic y de todas las dependencias necesarias para que funcione. Estas son:

- NodeJs
- Ionic
- Otras dependencias

Instalar NodeJs

En primer lugar tenemos que instalar el gestor de paquetes de NodeJs (*npm*) para poder instalar el propio Ionic y algunas otras dependencias que nos harán falta. Para instalarlo simplemente podemos descargarlo e instalarlo desde su Web:

<https://nodejs.org/en/>

O en Linux instalarlo usando el gestor de paquetes:

```
sudo apt-get install nodejs  
sudo apt-get install npm
```

Instalar Ionic

Para instalar la última versión de Ionic y Cordova(*) tenemos que ejecutar el siguiente comando:

```
$ sudo npm install -g cordova ionic
```

Nota: En Windows tenéis que quitar "sudo" del comando anterior.

(*) Nota: Como ya mencionamos en la introducción Ionic utiliza Cordova para acceder a las características nativas de los dispositivos móviles y también para poder compilar el código para cada plataforma como si fuera una app nativa.

Para comprobar si se ha instalado correctamente podemos escribir en un terminal el siguiente comando:

```
$ ionic
```

Esto nos tendría que mostrar un listado con todas las opciones disponibles del *cli* (intérprete de línea de comandos) de Ionic. Con esto ya tendríamos instalado tanto Ionic como Cordova, por lo que podríamos empezar a trabajar, pero antes de nada tendremos que revisar algunas dependencias más.

Otras dependencias

Hay algunas dependencias más que dependerán del sistema operativo que uséis y de las plataformas para las que queráis compilar vuestras aplicaciones. En la sección "Plataformas" veremos más información sobre esto.

Además también es posible que queráis usar Gulp, Bower o SASS, aunque su instalación es opcional sí que se recomiendan ya que os ayudarán mucho en el desarrollo de este tipo de aplicaciones:

```
$ sudo npm install -g gulp
$ sudo npm install -g bower
$ sudo gem install sass
```

¿Qué es Gulp? Gulp es un sistema de construcción que permite automatizar tareas comunes o repetitivas de desarrollo, tales como la minificación de código JavaScript, recarga del navegador, compresión de imágenes, validación de sintaxis de código y un sin fin de tareas más. Para mas información consultad: <http://gulpjs.com/>

¿Qué es Bower? Bower es un gestor de paquetes para el desarrollo Web. Es decir, cualquier librería de código abierto que puedas necesitar para el desarrollo de tu *front-end* lo podrás gestionar con Bower, él se encargará de instalar la librería y todas sus dependencias e incluso de las actualizaciones. Para mas información consultad: <http://bower.io/>

¿Qué es SASS? SASS es un metalenguaje de CSS que nos permite programar hojas de estilo usando variables, reglas CSS anidadas, *mixins* (facilitan la definición de estilos reutilizables), importación de hojas de estilos y muchas otras características. Ionic incluye los fuentes de sus hojas de estilos en SASS, así que podemos aprovechar para modificarlas usando este lenguaje. Para mas información consultad: <http://sass-lang.com/>

Conceptos básicos

En esta sección vamos a ver los conceptos básicos de la librería Ionic. En primer lugar crearemos nuestra primera aplicación y analizaremos la estructura que tienen los proyectos. A continuación veremos como generar una aplicación para diferentes plataformas, como Android o iOS. Y por último las opciones más importantes del interprete de línea de comandos (*cli*) de Ionic, para compilar y emular nuestros proyectos.

!Manos a la obra!

Crear nuestro primer proyecto

En esta sección vamos a ver los primeros pasos que tenemos que seguir para crear un proyecto:

1. Usar el *cli* de Ionic para crear el proyecto.
2. Añadir la plataforma para la que vamos a compilar.
3. Compilar y ejecutar el proyecto en un navegador, emulador o dispositivo real.

En las siguientes secciones se detallan cada uno de estos pasos.

1. Crear un nuevo proyecto

Ionic facilita la creación de proyectos usando su intérprete de línea de comandos (*cli*), mediante el cual podremos generar proyectos en "blanco" (en el que la primera pantalla esté vacía) para empezar a programar su contenido desde cero. Para crear un proyecto tenemos que escribir el siguiente comando en un terminal:

```
$ ionic start myApp blank
```

Al ejecutar este comando se creará una carpeta llamada "myApp" con el contenido del nuevo proyecto de Ionic. Además el código de la aplicación incluirá una primera pantalla vacía, en la cual podremos empezar a escribir nuestro código.

El *cli* de Ionic también permite generar proyectos con algo de contenido para no partir desde cero:

```
$ ionic start myApp tabs  
$ ionic start myApp sidemenu
```

Los cuales generarían respectivamente un proyecto de Ionic con la navegación mediante pestañas o con un menú lateral. En la imagen inferior se puede ver un ejemplo de estos tres tipos:



2. Añadir las plataformas destino

Como siguiente paso tendríamos que añadir las plataformas para las cuales queremos compilar. Ionic (con la ayuda de Cordova) permite generar código para multitud de plataformas, entre ellas están Android, iOS, Amazon Fire OS, Blackberry 10, navegador, Firefox OS, Ubuntu, WebOS, Windows Phone 8 y Windows. Las posibles plataformas para las que podemos generar dependerán del sistema operativo que utilicemos, por ejemplo, para iOS solo podremos compilar desde un Mac. En general soporta las siguientes combinaciones:

	Mac	Linux	Windows
iOS	x		
Amazon Fire OS	x	x	x
Android	x	x	x
BlackBerry 10	x	x	x
Navegador	x	x	x
Ubuntu		x	
Windows Phone 8			x
Windows			x
Firefox OS	x	x	x

Para añadir una plataforma para la cual queremos compilar nuestro proyecto usaremos el siguiente comando:

```
$ ionic platform add <nombre-de-la-plataforma>
```

Por ejemplo, para añadir la plataforma de compilación Android usaríamos:

```
$ ionic platform add android
```

Para ver un listado de las plataformas instaladas y las disponibles simplemente tenéis que ejecutar:

```
$ ionic platform
```

El código de las plataformas se añadirá dentro de la carpeta `platforms` del proyecto. Dentro de esta carpeta se copiará y compilará nuestro proyecto, generando las aplicaciones compiladas. Pero todo esto se realizará mediante comandos de Ionic, en ningún caso tenemos que modificar manualmente el código de esta carpeta.

Para más información podéis consultar:

https://cordova.apache.org/docs/en/edge/guide_platforms_android_index.md.html

3. Compilar y ejecutar el proyecto en un navegador, emulador o dispositivo real.

Para ejecutar el proyecto tenemos tres opciones: abrirlo en el navegador, emularlo o ejecutarlo en un terminal real. Según lo que queramos hacer nos puede interesar una opción u otra. Abrirlo en un navegador sería la opción más rápida y la más recomendable durante el desarrollo. Sin embargo, si usamos características nativas del dispositivo y queremos probarlas no podremos usar el navegador y tendremos que abrirlo en un emulador o dispositivo real.

3.1. Abrir en el navegador

Para abrir nuestro proyecto en un navegador para poder ver y depurar lo que vamos haciendo, simplemente tenemos que ejecutar el siguiente comando en un terminal en la raíz del proyecto:

```
$ ionic serve
```

Este terminal lo tenemos que dejar abierto mientras que estemos trabajando para que funcione el servidor. Al ejecutarlo nos mostrará el siguiente texto por pantalla:

```
Running dev server: http://localhost:8100
Ionic server commands, enter:
  restart or r to restart the client app from the root
  goto or g and a url to have the app navigate to the given url
  consolelogs or c to enable/disable console log output
  serverlogs or s to enable/disable server log output
  quit or q to shutdown the server and exit
```

Como se puede ver en las instrucciones el servidor se creara en la URL

`http://localhost:8100/` . Automáticamente se nos tendría que abrir una ventana del navegador en esta dirección, si no se abre también podemos poner la dirección manualmente. Con cada cambio que hagamos en el código del proyecto se recargará automáticamente el contenido del navegador. Si no funcionara por cualquier razón podéis escribir `restart` o `r` en el terminal.

Para parar el servidor simplemente tendremos que escribir `quit` o `q` en el terminal.

Como opción podemos añadir el parámetro `--lab` en la llamada (`$ ionic serve --lab`) que nos mostrará como quedaría la aplicación a la vez para Android y para iOS.

3.2. Emular o instalar en un dispositivo real

Ionic también permite emular o ejecutar el proyecto en un dispositivo real. Para esta opción primero tendremos que haber añadido una plataforma como vimos en el paso 2, en otro caso daría error.

Para emular un proyecto simplemente tendríamos que escribir en un terminal:

```
$ ionic emulate <PLATFORM>
```

Donde `<PLATFORM>` es la plataforma para la que queremos emular. Por ejemplo para emular en Android escribiríamos:

```
$ ionic emulate android
```

Y para instalar y ejecutar en un dispositivo real se hace de igual forma pero usando el comando `run` :

```
$ ionic run <PLATFORM>
```

En ambos casos podemos añadir la opción `--livereload` o `-l`, de esta forma tras cada cambio que hagamos en el código se recargará la emulación o la aplicación en el dispositivo. Por ejemplo:

```
$ ionic run -l ios
```

Esta opción todavía está en beta así que es normal que a veces de problemas y no funcione bien.

Además también podemos añadir en ambos casos la opción `--consolelogs` or `-c` para ver en el terminal la información que imprime por consola la aplicación.

Por último mencionar que si no queremos ejecutar el proyecto y solamente queremos compilarlo y generar el código destino de la plataforma podemos usar la opción `build` indicando también la plataforma de compilación:

```
$ ionic build <PLATFORM>
```

Si no indicamos la plataforma este comando compilaría el proyecto para todas las plataformas que tengamos instaladas.

Estructura de un proyecto

Al generar un nuevo proyecto de Ionic como hemos visto en el apartado anterior se nos creará una estructura predefinida de carpetas y ficheros que nos permitirán organizar el código de nuestro proyecto. A continuación vamos a ver para que vale cada una de estos elementos:

- `hooks/` - Esta carpeta se utiliza para añadir *scripts* que se ejecutarán cuando se produzcan determinados eventos, como por ejemplo antes o después, de la compilación, etc. En la propia carpeta se incluye un fichero con instrucciones para su utilización.
- `platforms/` - Contiene el código específico de las plataformas móviles para las cuales se va a compilar, como por ejemplo Android, iOS, etc. El código de esta carpeta es generado y **no se ha de modificar manualmente**.
- `plugins/` - Contiene los *plugins* o módulos instalados para nuestra aplicación, los cuales se utilizan para añadir funcionalidad como el acceso a las características nativas del móvil.
- `resources/` - Recursos específicos de las plataformas. En esta carpeta podremos colocar aquellos *assets* que sean únicos o dependientes de una plataforma en concreto.
- `scss/` - Código SCSS que será compilado a la carpeta `www/css/`
- `www/` - Contiene el código fuente principal de nuestra aplicación web: HTML, CSS, JavaScript, imágenes, etc. Esta carpeta es donde tendremos que desarrollar la aplicación web de forma centralizada y después utilizarla para la compilación para las distintas plataformas.
- `bower.json` - Listado de dependencias y paquetes de Bower.
- `config.xml` - Contiene la configuración de Cordova (o PhoneGap) con las opciones específicas para cada plataformas de compilación.
- `gulpfile.js` - Listado de tareas de Gulp.
- `ionic.config.json` - Configuración del proyecto de Ionic.
- `package.json` - Dependencias y paquetes de NodeJS.

Al crear un nuevo proyecto todas las carpetas se encontrarán vacías o incluirán algunos ficheros de ejemplo. Las carpetas `platforms` y `resources` no vienen por defecto sino que se crearán una vez añadamos la primera plataforma de compilación.

La carpeta `www` incluye a su vez una serie de subcarpetas y algo de código de ejemplo (este código dependerá de la plantilla con la que hayamos generado el proyecto: *blank*, *tabs* o *sidemenu*). Como hemos dicho, esta carpeta es donde tenemos que desarrollar el código

principal de nuestra aplicación así que tenemos que conocer bien su estructura:

- `css/` - Aquí colocaremos todas las hojas de estilo que se usen en la aplicación.
- `img/` - En esta carpeta almacenaremos las imágenes de nuestro proyecto.
- `js/` - Contendrá todo el código JavaScript de la aplicación.
- `lib/` - Aquí guardaremos todas las librerías que se usen en nuestro código. Por defecto ya viene incluido todo el código de la librería Ionic (Javascipts, CSS, etc.) para que lo podamos cargar desde nuestro proyecto.
- `templates/` - Esta carpeta viene preparada para almacenar las plantillas o vistas de la aplicación (en algunas versiones no se crea por defecto).
- `index.html` - Este es el fichero principal que se abrirá al cargar la aplicación. Aquí tendremos que cargar todo lo necesario y mostrar el contenido de la primera pantalla.

En las siguientes secciones analizaremos el código básico necesario que tenemos que incluir en el fichero `index.html` para crear una aplicación y los componentes principales que podemos utilizar.

Código básico

Al crear un nuevo proyecto de Ionic (usando la plantilla *blank*) se nos crea una aplicación de ejemplo con el contenido mínimo: una pantalla en blanco. Esta es la mejor opción para empezar a trabajar y añadirle los componentes que nosotros queramos o necesitemos.

Como ya hemos dicho todo el código de la aplicación lo encontraremos dentro de la carpeta `www`. Puesto que las apps de Ionic son básicamente páginas web, aquí dentro se incluirán carpetas y ficheros para almacenar código HTML, CSS y Javascript. El fichero más importante y principal es `index.html`, el cual iniciará la carga de la aplicación y definirá el contenido principal. A continuación se incluye el código básico que Ionic utiliza por defecto para este fichero:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1,
      user-scalable=no, width=device-width">
    <title></title>

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">

    <ion-pane>
      <ion-header-bar class="bar-stable">
        <h1 class="title">Ionic Blank Starter</h1>
      </ion-header-bar>
      <ion-content>
      </ion-content>
    </ion-pane>

  </body>
</html>
```

En primer lugar, en la sección `head` de este código, se cargan todas las dependencias necesarias:

- `lib/ionic/css/ionic.css` : Primero carga la hoja de estilo de Ionic, la cual aplicará los estilos visuales a los componentes.
- `css/style.css` : Esta hoja de estilo se incluye por defecto vacía para que escribamos en ella nuestros estilos CSS personalizados.
- `lib/ionic/js/ionic.bundle.js` : Librería JavaScript de Ionic y AngularJS.
- `cordova.js` : API JavaScript de Cordova.
- `js/app.js` : En este fichero se inicializa el módulo de Angular que gestionará la aplicación.

Si buscáis el fichero `cordova.js` dentro de la carpeta `www` no lo encontraréis. Esto es porque se incluirá después al compilar o emular.

A continuación en el `body` se define la aplicación de AngularJS a utilizar (`ng-app="starter"`). Este código asigna el módulo que hemos definido en `app.js` llamado `starter` para que gestione el contenido de la página. De esta forma ya podemos utilizar AngularJS para crear todo el contenido dinámico de la página.

Dentro del `body` se incluye un panel que contiene una cabecera (con el texto "*Ionic Blank Starter*") y una sección de contenido vacío.

En Ionic, la forma de incluir componentes es mediante el uso de clases CSS sobre etiquetas HTML o mediante el uso de directivas de AngularJS (permiten definir bloques de código HTML con estilos y funcionalidad asociada). En este caso se utilizan las directivas `ion-pane` , `ion-header-bar` y `ion-content` .

Ahora que ya sabemos por donde empezar en las siguientes secciones vamos a ver los componentes de interfaz de usuario que incluye Ionic y que podemos utilizar para crear nuestra aplicación.

Componentes

En este capítulo vamos a ver los componentes o elementos de interfaz que incluye la librería de Ionic. Estos componentes nos permitirán diseñar el aspecto gráfico de nuestra aplicación utilizando únicamente etiquetas HTML y algunos atributos para opciones de configuración.

Los principales componentes que podemos utilizar son:

- Área de contenidos
- Cabeceras
- Pie de página
- Botones y enlaces
- Listados
- Tarjetas
- Formularios
- Tabs o fichas
- Grid o rejillas
- Y algunas utilidades más

Mediante estos componentes podemos crear una aplicación estática completamente funcional, que tenga por ejemplo una pantalla con varias fichas, una cabecera y pie de página, y un listado de elementos, y todo esto escribiendo únicamente código HTML.

En las siguientes secciones veremos cada uno de estos componentes más en detalle junto con algunos ejemplos de uso.

Contenidos

Todo lo que escribamos dentro de la sección `<body>` del fichero `index.html` (que está en la carpeta " `www` " de nuestro proyecto) se mostrará como contenido visible de la aplicación. Dentro de esa sección podemos escribir cualquier etiqueta HTML o estilo CSS como si de una página normal se tratase. Por ejemplo, si escribimos el siguiente código:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1,
      maximum-scale=1, user-scalable=no, width=device-width">
    <title>My App</title>
    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">
    <ion-pane>
      <ion-content>
        <h1>I'm an H1!</h1>
        <h2>I'm an H2!</h2>
        <h3>I'm an H3!</h3>
        <h4>I'm an H4!</h4>
        <h5>I'm an H5!</h5>
        <h6>I'm an H6!</h6>
        <p>I'm a paragraph with a <a href="#">link</a>!</p>
      </ion-content>
    </ion-pane>
  </body>
</html>
```

Obtendríamos un resultado similar al siguiente:



Como se puede ver en el código, el contenido lo hemos puesto dentro de un panel, definido mediante la directiva `<ion-pane>`, que lo único que realiza es ajustar el tamaño al contenedor. Dentro de este panel podemos añadir lo que queramos: HTML o elementos definidos por Ionic como cabeceras, *footers*, listados, etc. El contenido principal tenemos que ponerlo dentro de la directiva `<ion-content>`, es importante usar esta etiqueta para que funcione el *scroll* correctamente ya que de otra forma los contenidos aparecerían bloqueados.

El área de contenido (la sección `<body>`) se mostrará en el móvil con un tamaño fijo, es decir, el usuario no podrá hacer zoom y solo podrá arrastrar con el dedo haciendo *scroll* de forma vertical. La directiva `<ion-content>` nos permite realizar ajustes sobre el contenido, como por ejemplo añadir un *padding* o margen interior con `<ion-content class="padding">`, o controlar el tipo de scroll. Para más información podéis consultar la documentación en: <http://ionicframework.com/docs/api/directive/ionContent/>

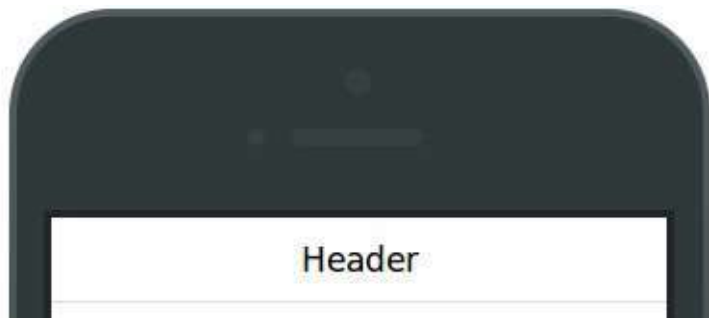
Resumiendo, el contenido principal de la aplicación tiene que ir dentro de la sección *body* entre las etiquetas `<ion-pane>` e `<ion-content>`. En caso de que sea un elemento que tenga que permanecer en una posición fija como una barra de cabecera o de pie de página

lo pondremos fuera de `<ion-content>` pero dentro de la etiqueta `<ion-pane>` .

En los siguientes apartados vamos a ver los componentes que incluye Ionic y que podemos añadir a esta sección de contenidos. Estos componentes, a diferencia de las etiquetas HTML normales, vienen con estilos predefinidos que emularán el aspecto de una aplicación nativa, para por ejemplo colocar una cabecera fija con un título en la parte superior o un listado de elementos bien formateado.

Cabeceras

Los elementos cabecera nos permiten poner una región fija en la parte superior de la aplicación que puede contener un título, botones a la izquierda o a la derecha además de otros componentes. A continuación se incluye un ejemplo:



El código para escribir esta cabecera sería el siguiente:

```
<ion-header-bar>
  <h1 class="title">Header</h1>
</ion-header-bar>
```

Como se puede ver la cabecera está formada por la directiva `<ion-header-bar>` y una etiqueta `<h1>` con la clase `title` para incluir el título en el centro.

Ionic incluye además varias clases CSS que podemos utilizar para modificar el color de la cabecera, estas son:

Color	Ejemplo
bar-light	
bar-stable	
bar-positive	
bar-calm	
bar-balanced	
bar-energized	
bar-assertive	
bar-royal	
bar-dark	

Estas clases CSS la tenemos que añadir a la definición de nuestra barra, por ejemplo, para que la barra aparezca en azul usaríamos el código:

```
<ion-header-bar class="bar-positive">
  <h1 class="title">bar-positive</h1>
</ion-header-bar>
```

O para que aparezca en un color oscuro:

```
<ion-header-bar class="bar-dark">
  <h1 class="title">bar-dark</h1>
</ion-header-bar>
```

Además también podemos usar el atributo `align-title` para indicar la alineación del título. A continuación se muestra un ejemplo completo:


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1,
      user-scalable=no, width=device-width">
    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">
    <ion-pane>
      <ion-header-bar class="bar-positive" align-title="center">
        <h1 class="title">Título</h1>
      </ion-header-bar>
      <ion-content class="padding">
        ...
      </ion-content>
    </ion-pane>
  </body>
</html>
```

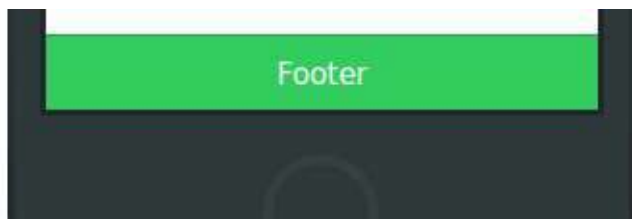
Como se puede ver la cabecera la tenemos que colocar fuera de las etiquetas `<ion-content>` pero dentro de la sección `<ion-pane>` .

Pie de página

Los pies de página nos permiten añadir una barra fija en la parte inferior de la pantalla (similar a las cabeceras) que pueden contener distintos tipos de contenidos. Para añadirlas tenemos que utilizar la directiva `<ion-footer-bar>` de la forma:

```
<ion-footer-bar class="bar-balanced">
  <h1 class="title">Footer</h1>
</ion-footer-bar>
```

Con lo que obtendríamos un resultado similar al siguiente:



Como se puede ver en el código para indicar el título del pie de página también tenemos que añadir una etiqueta `<h1>` con la clase " `title` ". Además podemos usar las clases CSS de color que vimos para las cabeceras, estas eran:

Color	Ejemplo
bar-light	<div>bar-light</div>
bar-stable	<div>bar-stable</div>
bar-positive	<div>bar-positive</div>
bar-calm	<div>bar-calm</div>
bar-balanced	<div>bar-balanced</div>
bar-energized	<div>bar-energized</div>
bar-assertive	<div>bar-assertive</div>
bar-royal	<div>bar-royal</div>
bar-dark	<div>bar-dark</div>

Por lo que al añadir en el código de ejemplo anterior la clase CSS " `bar-balanced` " daría como resultado un pie de página en color verde.

Ejemplo

A continuación se muestra un ejemplo de como podríamos crear una aplicación con una cabecera, un pie de página y un texto dentro del área de contenido:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1,
      user-scalable=no, width=device-width">
    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">
    <ion-pane>
      <ion-header-bar class="bar-positive">
        <h1 class="title">Cabecera</h1>
      </ion-header-bar>
      <ion-content class="padding">
        <h1>Mi primera aplicación</h1>
        <p>Texto de ejemplo.</p>
      </ion-content>
      <ion-footer-bar class="bar-balanced">
        <h1 class="title">Pie de página</h1>
      </ion-footer-bar>
    </ion-pane>
  </body>
</html>
```

Como se puede ver tanto el pie de página como la cabecera las tenemos que colocar fuera de las etiquetas `<ion-content>` pero dentro de la sección `<ion-pane>` .

Con este código obtendríamos un resultado similar al siguiente:

Cabecera

Mi primera aplicación

Texto de ejemplo.

Pie de página

Botones y enlaces

Los botones y enlaces son uno de los componentes más utilizados e incluso se podría decir que fundamentales en la programación de cualquier aplicación para un dispositivo móvil. Para utilizarlos simplemente tenemos que añadir la clase CSS `button` a un elemento tipo `button` o a un enlace tipo `a` normal (ambos casos se mostrarán visualmente de la misma forma), por ejemplo:

```
<button class="button">Apriétame!</button>
<a href="index.html" class="button">Apriétame!</a>
```

Los botones también tienen una serie de estilos CSS que nos permitirán cambiar su color. Los colores incluidos son los mismos que para las cabeceras y pies de página pero con el prefijo *"button"*. Estos son:



Para usarlos simplemente tenemos que añadir la clase CSS deseada de la forma:

```
<button class="button button-positive">Apriétame!</button>
<a href="index.html" class="button button-dark">Apriétame!</a>
```

Para crear un enlace que cargue una página distinta de la actual podemos hacerlo de forma normal, poniendo la dirección de la nueva página en el campo `href`, por ejemplo:

```
<a href="bibliografia.html" class="button">Ir a la bibliografía</a>
```

A continuación se van a explicar algunas opciones de configuración más que podemos usar con los botones.

Ancho del botón

Podemos variar el ancho del botón para que ocupe todo el ancho posible pero respetando el margen o *padding* definido (usando la clase `button-block`) o para que llegue hasta el borde de la pantalla eliminando los posibles márgenes (mediante la clase `button-full`). En el siguiente ejemplo se puede ver claramente la diferencia:



Para conseguir estos botones tendríamos que escribir el siguiente código:

```
<button class="button button-block button-positive">
  button-block
</button>

<button class="button button-full button-positive">
  button-full
</button>
```

Tamaño del botón

Podemos usar las clases `button-small` o `button-large` para crear botones más pequeños o más grandes, respectivamente:

```
<button class="button button-small button-assertive">
  Small Button
</button>

<button class="button button-large button-positive">
  Large Button
</button>
```

Botones con iconos

Podemos añadir imágenes a los botones de forma muy sencilla usando los iconos incluidos con Ionic (llamados Ionicons, ver sección de Iconos), o también podemos añadir iconos usando otra librería. Para añadir un icono a un botón se puede realizar de distintas formas:

- Dentro del propio botón o enlace añadiendo primero una de las siguientes clases: `icon`, `icon-left` o `icon-right`, para indicar la posición del icono, y después la clase que define el icono a utilizar (ver sección Iconos).
- También podemos añadir el icono añadiendo una etiqueta hija, dentro del botón o enlace. Esta opción es la recomendada si se usa una librería externa de iconos, como FontAwesome por ejemplo.

A continuación se incluyen algunos ejemplos de como añadir iconos:

```
<button class="button">
  <i class="icon ion-home"></i> Home
</button>

<button class="button icon-left ion-star button-positive">Favorites</button>

<a class="button icon-right ion-chevron-right button-calm">Learn More</a>

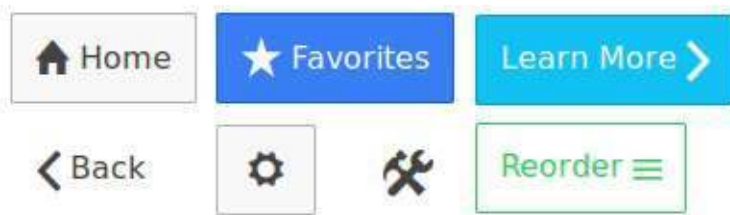
<a class="button icon-left ion-chevron-left button-clear button-dark">Back</a>

<button class="button icon ion-gear-a"></button>

<a class="button button-icon icon ion-settings"></a>

<a class="button button-outline icon-right ion-navicon button-balanced">Reorder</a>
```

Con lo que obtendríamos el siguiente resultado:



Al aplicar la clase `button-clear` sobre el botón hacemos que se muestre únicamente el texto y el icono, quitando el borde y el fondo. Si el botón únicamente tuviera icono entonces tenemos que usar la clase `button-icon` para que solo se vea el icono.

Botones en cabeceras o en pies de página

Para poner botones en una cabecera o pie de página simplemente tenemos que añadirlos antes o después del título, con lo cual provocaremos que el botón se coloque a la izquierda o a la derecha del título respectivamente, por ejemplo:

```
<ion-header-bar>
  <button class="button icon ion-navicon"></button>
  <h1 class="title">Header Buttons</h1>
  <button class="button">Edit</button>
</ion-header-bar>
```

Con lo que obtendríamos:



Si no hubiera título podemos usar la clase CSS `pull-right` para alinear el botón a la derecha de la barra:

```
<ion-footer-bar>
  <button class="button button-clear pull-right">Right</button>
</ion-footer-bar>
```

Al poner un botón en una cabecera o pie de página este tomará el estilo aplicado a la barra por defecto, por lo que no es necesario añadir ningún estilo más. Sin embargo si queremos le podemos aplicar algunas de las clases que hemos visto para colorear los botones.

También podemos borrar el borde y fondo del botón para que solamente se vea el icono del mismo. Para esto tenemos que añadir la clase `button-icon` en caso de que el botón solo tenga un icono o `button-clear` si tuviera texto, por ejemplo:

```
<ion-header-bar>
  <button class="button button-icon icon ion-navicon"></button>
  <h1 class="title">Header Buttons</h1>
  <button class="button button-clear button-positive">Edit</button>
</ion-header-bar>
```

Barra de botones

Los botones se puede agrupar de forma muy sencilla simplemente metiéndolos dentro de un `<div>` con la clase `button-bar`, por ejemplo:

```
<div class="button-bar">
  <a class="button">First</a>
  <a class="button">Second</a>
  <a class="button">Third</a>
</div>
```

Con lo que obtendríamos:



Estas barras de botones se puede utilizar también en cabeceras y en pies de página.

Listados

Los listados son un componente muy utilizado en las aplicaciones ya que son muy útiles para mostrar... eso es, una lista de elementos! Permiten incluir desde texto hasta otros elementos mas avanzados como iconos, imágenes, botones o *toggles*. Además se le podrán añadir funcionalidades más avanzadas como edición, arrastrar para reordenar, arrastrar para mostrar botones de edición o de borrado, estirar para actualizar, etc.

Para incluir un listado tenemos que usar la directiva `<ion-list>` y cada elemento del listado lo incluiremos con `<ion-item>`. Por ejemplo, para crear un listado con elementos tipo texto tendríamos que escribir el siguiente código:

```
<ion-list>
  <ion-item>
    Elemento 1
  </ion-item>
  <ion-item>
    Elemento 2
  </ion-item>
  <ion-item>
    ...
  </ion-item>
</ion-list>
```

El listado tendrá que ir dentro de la sección `<ion-content>` para que funcione el *scroll* correctamente, ya que en otro caso no se podría arrastrar. A continuación se incluye un ejemplo de una aplicación con listado y una cabecera:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1,
      maximum-scale=1, user-scalable=no, width=device-width">
    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">
    <ion-pane>
      <ion-header-bar class="bar-positive">
        <h1 class="title">Ejemplo de listado</h1>
      </ion-header-bar>
      <ion-content>
        <ion-list>
          <ion-item>Elemento 1</ion-item>
          <ion-item>Elemento 2</ion-item>
          <ion-item>...</ion-item>
        </ion-list>
      </ion-content>
    </ion-pane>
  </body>
</html>
```

Con lo que obtendríamos un resultado similar al siguiente:



Listados mediante enlaces

También podemos crear listados usando HTML y estilos CSS. Si no necesitamos usar ninguna opción adicional de la directiva para listados esta puede ser una buena opción. Simplemente tendremos que cambiar `<ion-list>` por `<div class="list">` y los elementos del listado por enlaces o campos `<div>` a los que se le aplique la clase `item`, por ejemplo:

```
<div class="list">
  <a class="item" href="enlace.html">
    Elemento 1
  </a>
  ...
</div>
```

Separadores

Podemos crear separadores dentro de una lista para organizar o agrupar mejor los elementos de la misma. Para crear un separador simplemente tenemos que añadirle la clase `item-divider`. Estos elementos se mostrarán con un estilo distinto al del resto de elemento. A continuación se incluye un ejemplo:

```
<ion-list>
  <ion-item class="item-divider">
    Cabecera
  </ion-item>
  <ion-item>Elemento 1</ion-item>
  <ion-item>Elemento 2</ion-item>
  ...
</ion-list>
```

Con lo que obtendríamos un resultado similar al siguiente:

Ejemplo de listado	
Cabecera	
Elemento 1	
Elemento 2	
Cabecera	
Elemento 3	
Elemento 4	

Listados con iconos

Podemos poner iconos en la parte izquierda o derecha de los elementos de un listado simplemente usando unas clases CSS predefinidas y los iconos incluidos (Ionicons) u otras de otra librería de iconos.

Las clases CSS para la alineación del icono se tienen que aplicar sobre la etiqueta de apertura del elemento de la lista. Si queremos poner un icono en la izquierda tenemos que añadir la clase CSS `item-icon-left` y para alinearlo a la derecha la clase `item-icon-right`. Para poner dos iconos, uno alineado a cada lado tenemos que aplicar ambas clases.

A continuación se incluye un ejemplo en el que el primer elemento solo tiene un icono alineado a la izquierda, el segundo tiene a ambos lados, el tercero además tiene una nota a la derecha y en el último se incluye un *badge*:

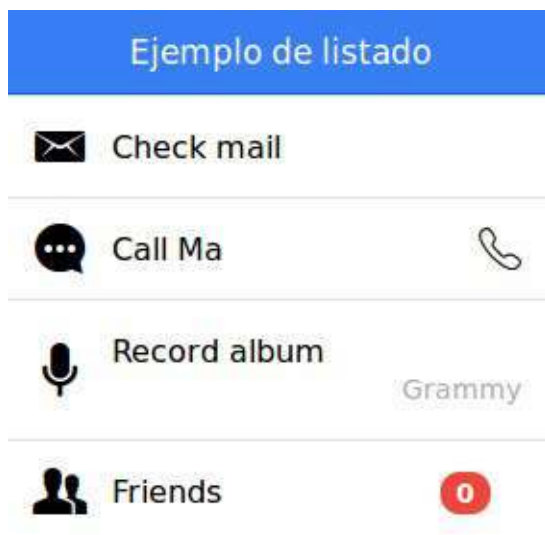
```
<ion-list>
  <ion-item class="item-icon-left">
    <i class="icon ion-email"></i>
    Check mail
  </ion-item>

  <ion-item class="item-icon-left item-icon-right">
    <i class="icon ion-chatbubble-working"></i>
    Call Ma
    <i class="icon ion-ios-telephone-outline"></i>
  </ion-item>

  <ion-item class="item-icon-left">
    <i class="icon ion-mic-a"></i>
    Record album
    <span class="item-note">
      Grammy
    </span>
  </ion-item>

  <ion-item class="item-icon-left">
    <i class="icon ion-person-stalker"></i>
    Friends
    <span class="badge badge-assertive">0</span>
  </ion-item>
</ion-list>
```

Este código se mostraría de la forma:

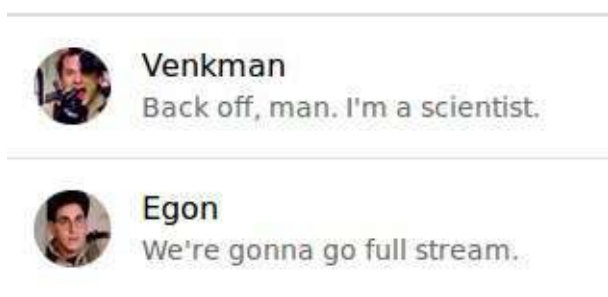


Elementos con imágenes tipo *Avatars*

Además de los iconos podemos mostrar imágenes un poco más grandes que se verán redondeadas, imitando el estilo tipo *avatar*. Para esto simplemente tenemos que añadir la clase CSS `item-avatar` siguiendo la siguiente estructura:

```
<ion-list>
  <ion-item class="item-avatar">
    
    <h2>Venkman</h2>
    <p>Back off, man. I'm a scientist.</p>
  </ion-item>
  ...
</ion-list>
```

Con lo que obtendríamos un resultado similar al de la siguiente imagen:



Elementos con imágenes en miniatura (*thumbnails*)

Además de iconos y avatares también podemos incluir imágenes un poco mas grandes tipo *thumbnail*. Para esto tenemos que usar la clase `item-thumbnail-left` si queremos colocarla a la izquierda o la clase `item-thumbnail-right` para ponerla a la derecha. A continuación se incluye un ejemplo de uso:

```
<ion-list>
  <ion-item class="item-thumbnail-left">
    
    <h2>Weezer</h2>
    <p>Blue Album</p>
  </ion-item>
  ...
</ion-list>
```

Con lo que obtendríamos un resultado similar al de la siguiente imagen:



Weezer
Blue Album




Smashing Pumpkins
Siamese Dream

Tarjetas

Las tarjetas (*cards*) son otro elemento ampliamente usado para mostrar y organizar la información. Para utilizarlas simplemente tenemos que escribir un bloque de código como el siguiente:

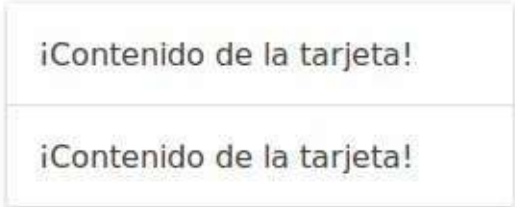
```
<div class="card">
  <div class="item item-text-wrap">
    ¡Contenido de la tarjeta!
  </div>
</div>
```

Con lo que obtendríamos el siguiente resultado:



¡Contenido de la tarjeta!

Dentro de la sección interior (`item`) podemos incluir todo el código que queramos: texto, otras etiquetas HTML u otros componentes de Ionic. Si ponemos varias secciones `item` simplemente se verán apiladas una después de la otra:



¡Contenido de la tarjeta!

¡Contenido de la tarjeta!

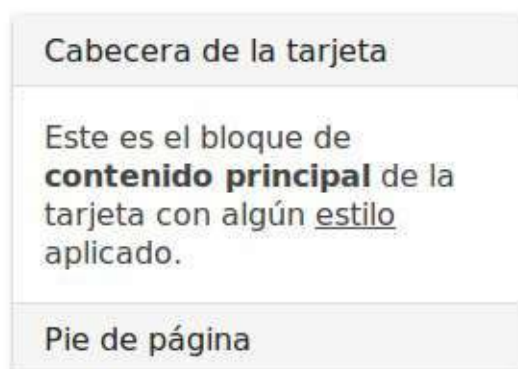
A continuación vamos a ver algunos elementos más que podemos usar con las tarjetas.

Cabeceras y pies de tarjeta

Las tarjetas se puede personalizar de forma similar a como si fueran una pantalla normal. Por ejemplo, podemos añadirle de forma muy sencilla una cabecera y un pie de página dentro de la misma tarjeta. Para esto simplemente tenemos que utilizar la clase CSS `item-divider` antes o después del bloque de contenido:

```
<div class="card">
  <div class="item item-divider">
    Cabecera de la tarjeta
  </div>
  <div class="item item-text-wrap">
    Este es el bloque de <b>contenido principal</b>
    de la tarjeta con algún <u>estilo</u> aplicado.
  </div>
  <div class="item item-divider">
    Pie de página
  </div>
</div>
```

Con lo que obtendríamos el siguiente resultado:



Listas de elementos

Los elementos de la tarjeta también pueden ser enlaces (`<a>`) sobre los que aplicamos la clase CSS `item` . Estos bloques se pueden apilar unos debajo de otros creando una tarjeta con diferentes secciones, por ejemplo una cabecera, una sección de contenido principal y por último enlaces. En estos bloques también se pueden incluir iconos, pero para esto tenemos que aplicar la clase CSS `item-icon-left` . A continuación se incluye un ejemplo:

```
<div class="card">
  <div class="item item-divider">
    Tarjeta empresa
  </div>
  <div class="item item-text-wrap">
    Información detallada de la empresa...
  </div>
  <a href="#" class="item item-icon-left">
    <i class="icon ion-ios-telephone"></i>
    Contactar con la empresa
  </a>
  <a href="#" class="item item-icon-left">
    <i class="icon ion-card"></i>
    Ver información de pago
  </a>
</div>
```

Con este código obtendríamos una tarjeta similar a la siguiente:



Formularios

En Ionic, para crear la disposición de los elementos de un formulario, vamos a utilizar también un elemento lista en el que cada fila será un *input* del formulario. Esto nos permitirá además aprovechar todas las opciones que vimos para las listas, como por ejemplo añadir cabeceras de secciones para agrupar contenidos.

Both `item-input` and `item` is then used to designate each individual input field and it's associated label.

Ionic prefers to create the `item-input` out of the element so that when any part of the row is tapped then the underlying input receives focus.

Additionally, developers should be aware of the various HTML5 Input types so users will be presented with the appropriate virtual keyboard.

<http://ionicframework.com/html5-input-types/>

Text Input: Placeholder Labels

In the example, it'll default to 100% width (no borders on the left and right), and uses the placeholder attribute to simulate the input's label. Then the user begins to enter text into the input the placeholder label will be hidden. Notice how

can also be used as a multi-line text input.

```
<div class="list">
  <label class="item item-input">
    <input type="text" placeholder="First Name">
  </label>
  <label class="item item-input">
    <input type="text" placeholder="Last Name">
  </label>
  <label class="item item-input">
    <textarea placeholder="Comments"></textarea>
  </label>
</div>
```

Text Input: Inline Labels

Use `input-label` to place a label to the left of the input element. When the user enters text the label does not hide. Note that there's nothing stopping you from also using a placeholder label too.

```
<div class="list">
  <label class="item item-input">
    <span class="input-label">Username</span>
    <input type="text">
  </label>
  <label class="item item-input">
    <span class="input-label">Password</span>
    <input type="password">
  </label>
</div>
```

Checkbox

Checkboxes allow the user to select a number of items in a set of choices. A good use for a checkbox list would be a filter list to apply multiple filters to a search.

Checkboxes can also have colors assigned to them, such as `checkbox-assertive` to assign the assertive color.

```
<ion-list>
  <ion-checkbox ng-model="filter.blue">Red</ion-checkbox>
  <ion-checkbox ng-model="filter.yellow">Yellow</ion-checkbox>
  <ion-checkbox ng-model="filter.pink">Pink</ion-checkbox>
</ion-list>
```

Radio buttons

Radio buttons let the user select one option in a set of options, unlike a checkbox which allows for multiple selections.

```
<ion-list>
  <ion-radio ng-model="choice" ng-value="'A'">Choose A</ion-radio>
  <ion-radio ng-model="choice" ng-value="'B'">Choose B</ion-radio>
</ion-list>
```

Botones de activación (*toggle*)

A toggle technically is the same thing as an HTML checkbox input, except it looks different and is easier to use on a touch device. Ionic prefers to wrap the checkbox input with the `ion-toggle` component in order to make the entire toggle easy to tap or drag.

Toggles can also have colors assigned to them, such as `toggle-assertive` to assign the assertive color.

```
<label class="toggle">
  <input type="checkbox">
  <div class="track">
    <div class="handle"></div>
  </div>
</label>
```

This is an example of multiple toggles within a list. Note the `item-toggle` class was added along side `item` for each item.

```
<ul class="list">
  <li class="item item-toggle">
    HTML5
    <label class="toggle toggle-assertive">
      <input type="checkbox">
      <div class="track">
        <div class="handle"></div>
      </div>
    </label>
  </li>
  ...
</ul>
```

Select o Desplegables

Ionic's select is styled so its appearance is prettied up relative to the browser's default style. However, when the select elements is opened, the default behavior on how to select one of the options is still managed by the browser.

Each platform's user-interface will be different as the user is selecting an option. For example, on a desktop browser you'll see the traditional drop down interface, whereas Android often has a radio-button list popup, and iOS has a custom scroller covering the bottom half of the window.

```
<div class="list">
  <label class="item item-input item-select">
    <div class="input-label">
      Lightsaber
    </div>
    <select>
      <option>Blue</option>
      <option selected>Green</option>
      <option>Red</option>
    </select>
  </label>
</div>
```

Selectores de rango

This is a Range. Ranges can be themed to any default Ionic color, and used in various other elements such as a list item or card.

```
<div class="item range">
  <i class="icon ion-volume-low"></i>
  <input type="range" name="volume">
  <i class="icon ion-volume-high"></i>
</div>

<div class="list">
  <div class="item range range-positive">
    <i class="icon ion-ios-sunny-outline"></i>
    <input type="range" name="volume" min="0" max="100" value="33">
    <i class="icon ion-ios-sunny"></i>
  </div>
</div>
```

Más información

- Input types: <http://ionicframework.com/html5-input-types/>

Iconos

Como ya habéis visto en Ionic se pueden incluir iconos dentro de multitud de componentes, como botones, listados, tarjetas, cabeceras, etc.

Para incluir uno de estos iconos se realiza simplemente añadiendo unas clases CSS que estan definidas en la librería de iconos de Ionic. Estas clases se pueden añadir sobre el mismo componente o de forma separada, para crear el icono por separado, por ejemplo:

```
<i class="icon ion-star"></i>
```

Ionic incluye la librería Ionicons con un montón de iconos que podemos usar. Para ver la lista completa podéis acceder a la dirección:

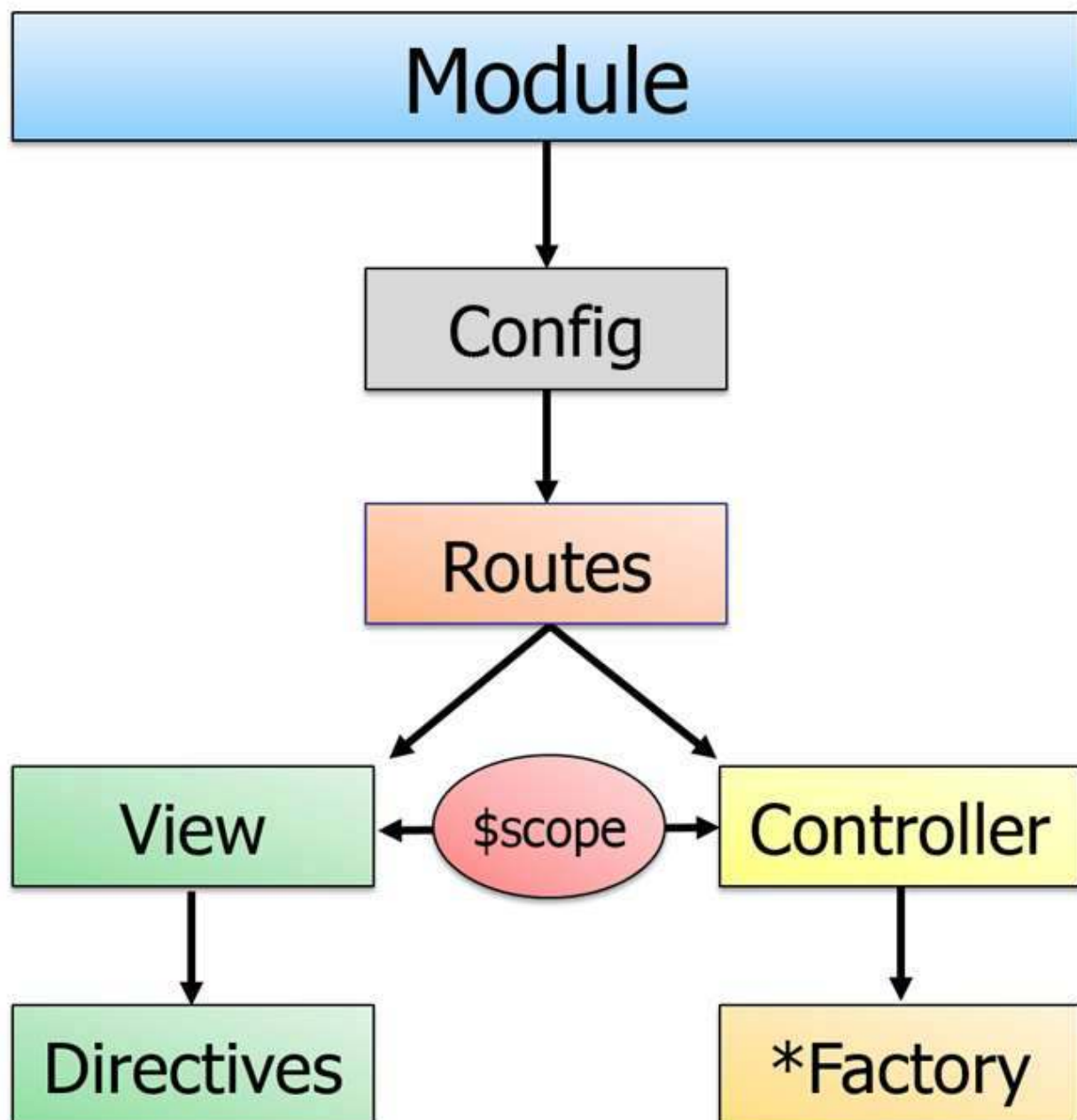
<http://ionicons.com/>

En esta página, al pulsar sobre cada icono os aparecerá la clase CSS que tenéis que utilizar.

Arquitectura de una aplicación con Ionic

Ionic, al estar basado en Angular, utiliza el patrón conocido como Vista-Controlador (*View-Controller*) que fue popularizado por *frameworks* como *Cocoa Touch*. En este tipo de patrón las diferentes secciones de la interfaz se pueden dividir en distintas vistas hijas o incluso podrían ser vistas hijas que contengan a su vez otras vistas hijas. Los controladores están asociados a estas vistas y se encargan de proporcionar los datos necesarios y la funcionalidad de los diferentes elementos.

En la siguiente imagen se puede ver un esquema de la arquitectura completa que sigue Ionic y Angular:



En la arquitectura de una aplicación intervienen muchos tipos de componentes además de las Vistas y Controladores, iremos viendo cada uno de ellos poco a poco, pero principalmente los que nos interesan son las Vistas, Controladores, Servicios o Factorías, y la Configuración y Rutas.

Intuitivamente, la tarea de cada uno de estos componentes en una aplicación con Ionic es la siguiente:

- Los controladores obtienen los datos de uno o varios Servicios o Factorías y lo envían a una vista o *template* a través de la variable `$scope`.
- Las vistas o *templates* contienen la descripción visual de una pantalla (o de una parte de una pantalla) y obtienen los datos a mostrar de la variable `$scope`.
- La configuración y las rutas de la aplicación permiten enlazar los controladores con las vistas o *templates* correspondientes.
- Las directivas permiten crear y usar componentes con aspecto y comportamiento personalizado.

Inicializando la aplicación

Como ya hemos visto en secciones anteriores, el punto de inicio de una aplicación Ionic es el fichero `index.html`, el cual como mínimo deberá tener el siguiente contenido:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1,
      user-scalable=no, width=device-width">
    <title></title>

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">

    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">

    <ion-nav-bar class="bar-positive">
      <ion-nav-back-button class="button-clear">
        <i class="ion-chevron-left padding-left"></i>
      </ion-nav-back-button>
    </ion-nav-bar>
    <ion-nav-view></ion-nav-view>

  </body>
</html>
```

En este código en primer lugar en la cabecera se cargan las dependencias de la aplicación: hojas de estilo, librería JavaScript de Ionic (la cual incorpora Angular), librería de Cordova y los ficheros JavaScript de nuestra aplicación (`js/app.js`).

A continuación en el `body` se indica el módulo de Angular a utilizar con `ng-app="starter"` . Esta línea inicia la carga del módulo llamado `starter` que estará definido en el fichero `js/app.js` de la forma:

```
angular.module('starter', ['ionic'])
```

Esta es la forma de crear un módulo con Angular, como primer parámetro indicamos el nombre del módulo (`starter`) y a continuación las dependencias, que en este caso solo se carga la librería de `ionic` . A continuación podremos indicar la configuración, los controladores y servicios que componen el módulo de la forma:

```
angular.module('starter', ['ionic'])

.config( /* ... */ )

.controller( /* ... */ )

.factory( /* ... */ )
```

En las siguientes secciones veremos cada uno de estos apartados por separado y por último se incluye un ejemplo completo.

Configuración y rutas

La configuración nos permite establecer las rutas o estados (`states`) que va a tener la aplicación y enlazar cada uno de ellos con una ruta, un controlador y un *template* (la vista). Además se tendrá que especificar también una ruta inicial o por defecto.

A continuación se incluye un ejemplo de como especificar la configuración de un módulo:

```
.config(function($stateProvider, $urlRouterProvider) {

    $stateProvider
        .state('home', {
            url: '',
            templateUrl: 'templates/home.html',
            controller: 'homeCtrl'
        })
        .state('page', {
            url: '/page',
            templateUrl: 'templates/page.html',
            controller: 'pageCtrl'
        });

    // Página por defecto
    $urlRouterProvider.otherwise('');
});
```

En este ejemplo se establecen dos rutas o estados. La primera llamada `home` , que tendrá la URL vacía y usará el *template* `home.html` y el controlador `homeCtrl` . Y la segunda ruta llamada `page` que tendrá la URL `/page` , usará el *template* `page.html` y el controlador `pageCtrl` . Además se establece que por defecto (en la primera llamada y cuando la ruta no exista) se usará la ruta vacía, que se corresponde con el estado `home` .

Si decimos que incluya una plantilla que no existe o nos equivocamos en el nombre del fichero html no mostrará ningún error, simplemente no funcionará.

Al realizar una consulta a la aplicación en primer lugar se creará el módulo con esta configuración y a continuación se cargará la ruta solicitada o la ruta por defecto. Si por ejemplo se solicita el estado `home` se llamará al controlador indicado para preparar los datos y a continuación se cargará el *template* `home.html` , se sustituirán los valores de la plantilla con los indicados en el controlador y por último se mostrará la vista al usuario.

Enlaces

Para crear un enlace que nos lleve a un estado indicado en la configuración simplemente tenemos que usar el atributo `ui-sref` con el nombre del estado deseado. Por ejemplo, para volver al *home* podríamos hacer:

```
<a ui-sref="home">Volver al inicio</a>
```

Si ponemos un enlace a una dirección que no existe o a un state que no existe no mostrará un error. Al pulsar intentará cambiar de pantalla pero volverá a mostrarse la misma donde estaba.

Rutas con parámetros

También podemos generar rutas con parámetros para pasar valores entre vistas. Por ejemplo, podemos tener un listado de usuarios y al pulsar sobre un elemento de la lista abrir una nueva pantalla con la vista detalle del usuario. Para esto es necesario que el enlace indique el índice del elemento pulsado para así poder abrir la vista con los datos del usuario correspondiente.

Para añadir parámetros a las rutas simplemente tenemos que indicarlos anteponiendo dos puntos (`:`) al nombre del parámetro en la *url*, por ejemplo:

```
.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
    .state('users', {
      url: '/users'
    })
    .state('userDetail', {
      url: '/user/:userId'
    })
    .state('userPost', {
      url: '/user/:userId/:postId'
    });
});
```

En el ejemplo anterior se definen tres rutas o estados. La primera (`users`) no tiene ningún parámetro y la usaremos para mostrar toda la lista de usuarios. La segunda (`userDetail`) recibe un parámetro (`userId`) a partir del cual podremos obtener los datos del usuario solicitado. Y el último estado (`userPost`) recibe dos parámetros para indicar un artículo de un usuario.

También podemos definir los parámetros de las rutas usando llaves en lugar de dos puntos. Por ejemplo, para el último estado del ejemplo anterior podríamos haber puesto `/user/{userId}/{postId}`.

Para generar un enlace a una ruta con parámetros usaremos también el atributo `ui-sref` y simplemente tenemos que añadir al nombre del estado un objeto entre paréntesis con los valores. A continuación se muestra un ejemplo:

```
<a ui-sref="users">Ver todos los usuarios</a>
<a ui-sref="userDetail({ userId: id })">Detalles del usuario</a>
<a ui-sref="userPost({ userId: id, postId: pid })">Detalle artículo</a>
```

Los nombres de las propiedades del objeto tienen que coincidir con los nombres que hayamos puesto a los parámetros de la ruta. Por ejemplo, en `userDetail({ userId: id })`, el nombre `userId` coincide con el de la ruta `/user/:userId`.

En el siguiente apartado sobre controladores veremos como recoger estos valores desde un controlador.

Más información

Para más información podéis consultar:

- <https://github.com/angular-ui/ui-router/wiki>
- <https://github.com/angular-ui/ui-router/wiki/URL-Routing#stateparams-service>
- <https://scotch.io/tutorials/3-simple-tips-for-using-ui-router>

Controladores

Los controladores podríamos decir que son el equivalente al cerebro de la aplicación, ya que son los que gestionan el flujo de los datos. Al mostrar una página de la aplicación en primer lugar se llama a un controlador, este controlador usará una vista (o *template*) para generar dicha página y además cargará los datos necesarios mediante servicios (*services* o *factores*, que veremos después). El controlador envía los datos necesarios a la plantilla a través de la variable `$scope`, de esta forma desde la vista solo tenemos que mostrar los datos recibidos dándoles el formato adecuado.

Por ejemplo, al acceder a la página `#miSuperPagina` se llamará automáticamente al controlador que tendrá como nombre `miSuperPaginaCtrl`. Este controlador está configurado para usar la plantilla llamada `superpagina.html` con el siguiente contenido:

```
<ion-view title="About">
  <ion-content>
    Mi nombre es {{user.name}}.
  </ion-content>
</ion-view>
```

Para definir el controlador asociado `miSuperPaginaCtrl` lo podríamos realizar de la forma:

```
.controller('miSuperPaginaCtrl', function($scope, $stateParams, superService) {
  $scope.user = superService.getUser();
})
```

Los parámetros que recibe la función del controlador serán inyectados por Angular en la llamada. Por lo tanto podemos usar estos parámetros para cargar servicios o clases que necesitemos. En este ejemplo se inyecta la variable `$scope` que como hemos visto nos permite pasar datos a la vista, también se inyecta `$stateParams` que sirve para recuperar los parámetros de llamada a la ruta o pantalla actual, y por último se carga `superService` que es un servicio que hemos creado para acceder los datos del usuario.

Si utilizamos alguna clase y se nos olvida inyectarla en los parámetros, como por ejemplo si estamos usando `$stateParams` pero no lo hemos puesto para que lo incluya, nos lanzará una excepción indicando que `$stateParams` no está definido.

El método `getUser()` del servicio `superService` devolverá un objeto con el siguiente formato:

```
var user = {  
  name: "Juan"  
}
```

Al llamar a esta página en primer lugar se ejecutará el controlador asociado, el cual asignará al `$scope` la variable `user` con los datos del usuario. A continuación se procesará la plantilla, la cual accederá a la variable `user.name` para obtener el nombre del usuario y sustituirlo. Finalmente obtendríamos el siguiente resultado:

```
<ion-view title="About">  
  <ion-content>  
    Mi nombre es Juan.  
  </ion-content>  
</ion-view>
```

Parámetros

La clase `$stateParams` nos permite recoger el valor de los parámetros de entrada proporcionados a una ruta. Por lo tanto, en el controlador tenemos que añadir esta clase a sus argumentos (para que Angular la inyecte) y después simplemente accederemos a los parámetros como si fueran propiedades de este objeto. Por ejemplo, si hemos definido en la configuración una ruta para mostrar la vista detalle de un usuario:

```
.config(function($stateProvider, $urlRouterProvider) {  
  $stateProvider  
    .state('userDetail', {  
      url: '/user/:userId'  
    });  
})
```

El parámetro `userId` que recibe esta ruta lo podríamos recoger en el controlador de la forma:

```
.controller('miSuperPaginaCtrl', function($scope, $stateParams, superService) {  
  var userId = $stateParams.userId;  
  
  // Usamos userId para obtener los datos del usuario  
  $scope.user = superService.getUser( userId );  
})
```

De esta forma podemos acceder a todos los parámetros pasados a través de la ruta. Tenemos que tener cuidado de que el nombre de variable sea el mismo y que esté definido, de otra forma obtendríamos el valor *undefined*.

Más información

Para más información podéis consultar la página:

- <https://docs.angularjs.org/guide/controller>
- <http://mcgivery.com/controllers-ionicangular/>

Vistas

Las vistas en una aplicación de Ionic se refieren al concepto de "*templates*" según la librería Angular. Por este motivo todas las vistas se guardarán dentro de la carpeta " `templates/` " en ficheros separados con extensión `.html` .

Las vistas contendrán la descripción de la parte gráfica de una pantalla, por lo que serán básicamente código HTML mezclado con CSS y también podremos usar otras directivas o componentes de Ionic, como botones, barras, etc. A continuación se incluye un ejemplo de una vista que guardaremos en el fichero `/templates/about.html` :

```
<ion-view title="About">
  <ion-content>
    Contenido de la vista
  </ion-content>
</ion-view>
```

Las vistas tienen que ir encerradas dentro de una sección `<ion-view>` a la que se le puede asignar opcionalmente un título. Este título se utiliza desde otras directivas como `<ion-nav-bar>` para obtener el texto de la barra de título.

Contenedor o página principal

Pero ¿dónde se muestran estas vistas y el título indicado? Al inicio de esta sección, al incluir el código del fichero `index.html` , dentro de la sección `<body>` añadimos dos directivas para esto:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- ... -->
  </head>
  <body ng-app="starter">

    <ion-nav-bar class="bar-positive">
      <ion-nav-back-button class="button-clear">
        <i class="ion-chevron-left padding-left"></i>
      </ion-nav-back-button>
    </ion-nav-bar>
    <ion-nav-view></ion-nav-view>

  </body>
</html>
```

Como se puede intuir, la directiva `<ion-nav-bar>` mostrará la barra de navegación y además incluirá el título indicado en la vista. Y la directiva `<ion-nav-view>` es donde se incluirá el contenido de la plantilla de la vista actual.

El botón de la barra de título indicado mediante la directiva `ion-nav-back-button` permitirá volver a la página anterior y solo se mostrará cuando se pueda pulsar.

Sustitución de variables

Dentro de la vista o plantilla podemos acceder a los datos proporcionados por el controlador. Para mostrarlos tenemos que usar una plantilla con el formato `{{nombre_variable}}` que se sustituirá por el valor de dicha variable. Por ejemplo:

```
<ion-view title="About">
  <ion-content>
    Mi nombre es {{name}}
  </ion-content>
</ion-view>
```

donde `{{name}}` es una variable que se ha asignado al `$scope` en el controlador de la forma:

```
$scope.name = "Juan";
```

Por lo que al mostrar la vista y una vez sustituidas las variables se podrá leer el texto "Mi nombre es Juan".

Es importante que nos fijemos que en la vista tenemos que usar directamente el nombre de la variable mientras que en el controlador las tenemos que añadir al objeto

```
$scope .
```

Bucles

Con Angular es posible crear bucles directamente en la plantilla para repetir un trozo de código. Para esto usaremos el atributo `ng-repeat` en la etiqueta que queremos que se repita. Es importante que nos fijemos en que la etiqueta la hemos de colocar en el mismo atributo a repetir y no fuera. Por ejemplo, para crear una lista el elemento a repetir sería la etiqueta `` :

```
<ul>
  <li ng-repeat="autor in autores">
    {{autor.nombre}} {{autor.apellidos}}
  </li>
</ul>
```

En este código creamos un bucle a partir de la variable `autores` , la cual contendrá un array de objetos con los datos de los autores. Esta variable se habrá asignado al `$scope` en el controlador correspondiente de la forma:

```
$scope.autores = [
  { nombre: "Juan", apellidos: "García" },
  { nombre: "Laura", apellidos: "Pérez" }
]
```

Después de procesar el bucle el código HTML que obtendríamos sería el siguiente:

```
<ul>
  <li>
    Juan García
  </li>
  <li>
    Laura Pérez
  </li>
</ul>
```

Condiciones

Además de sustituir variables y crear bucles Angular también permite crear condiciones mediante el uso de `ng-if/ng-show`. Ambos atributos realizarán la misma acción: mostrar u ocultar el contenido de la etiqueta en la que se encuentren dependiendo de la condición. Entonces, ¿cuál es la diferencia? Pues muy sencilla: con `ng-if` la etiqueta se eliminará del DOM si la condición es falsa, mientras que con `ng-show` la etiqueta simplemente se ocultará (pero permanecerá en el DOM).

Por ejemplo, para mostrar un mensaje si la lista de autores estuviera vacía simplemente tendríamos que hacer:

```
<div ng-show="!autores.length">
  La lista está vacía
</div>
```

El valor pasado a `ng-if/ng-show` puede ser una variable, una expresión o una llamada a una función, por ejemplo:

```
<div ng-show="mostrarAviso()">
  Aviso de ejemplo!
</div>
```

Y en el controlador podríamos añadir dicho método al `$scope` simplemente haciendo:

```
$scope.mostrarAviso = function(){
  return false;
}
```

Más información

Para más información podéis consultar:

- <https://docs.angularjs.org/guide/templates>
- <https://docs.angularjs.org/api/ng/directive>
- <http://mcgivery.com/creating-views-with-ionic/>

Servicios (*Factories/Services*)

La capa de datos en una aplicación Ionic o Angular se encarga de proporcionar, como su propio nombre indica, los datos desde un almacenamiento local o desde un servicio externo. Esta capa de datos la proporcionan las clases conocidas como *Servicios* o *Factories*. Ambas clases son muy similares y nos referiremos a ellas como capa o clase de datos. La diferencia estriba en el valor devuelto, los servicios siempre tienen que devolver un objeto (ya que al inyectarlos se les llamará con `new`), mientras que los *Factories* son más versátiles y podrán devolver lo que queramos ya que simplemente se ejecutará la función que lo define.

Como hemos visto el proceso seguido en una petición es el siguiente: El controlador solicita los datos a la capa de datos para prepararlos y pasárselos a la vista. La capa de datos normalmente definirá una serie de métodos para el acceso a los datos. A continuación se incluye un ejemplo sencillo de una capa de datos:

```
.factory('superService', function() {  
  return {  
    getUser: function() {  
      var user = { name: "Juan" };  
      return user;  
    },  
  }  
})
```

En este caso el servicio `superService` define un único método `getUser()` que devolverá un objeto con el nombre del usuario. Siguiendo este esquema podemos añadir todos los métodos que queramos al servicio y definir una API de consulta.

Para usar un servicio desde un controlador en primer lugar hay que solicitarlo en los parámetros para que Angular lo inyecte. Posteriormente, desde dentro del código de la función ya lo podemos usar y acceder a los valores o funciones que hayamos definido en la clase de datos:

```
.controller('MiSuperControladorCtrl', function($scope, superService) {  
  $scope.user = superService.getUser();  
})
```

Consultas asíncronas

En el ejemplo anterior los datos estaban guardados en una variable, pero lo más común es que tengamos que consultar esos datos desde un servicio remoto o desde una base de datos. Si utilizamos un servicio que pueda tardar en devolver la respuesta tendremos que trabajar de forma asíncrona para no bloquear la aplicación.

Al realizar una petición asíncrona la función del servicio no podrá devolver el valor directamente, sino que usará la función `then` para ayudarnos con la respuesta asíncrona. Por ejemplo, para hacer una consulta HTTP a una API externa y devolver la respuesta podríamos escribir el siguiente código en el servicio:

```
.factory('superService', function($http) {
  return {
    getUser: function() {
      return $http.get("url-de-consulta").then(function(response) {
        //...
        return response;
      });
    },
  },
})
```

Dentro de la función `then` podemos procesar los datos obtenidos y después devolver la respuesta. Para usar este servicio desde un controlador lo realizaremos como en el ejemplo anterior: solicitamos que el servicio se inyecte en los parámetros, llamamos al método de forma normal `superService.getUser()` pero la respuesta la tenemos que procesar con la función `then`. Esta función lo que hará es **quedarse esperando** a recibir la respuesta y una vez obtenida se ejecutará la función asignada como primer parámetro:

```
.controller('MiSuperControladorCtrl', function($scope, superService) {
  superService.getUser().then(function(response){
    $scope.user = response;
  });
})
```

Además podemos asignar un segundo parámetro a la función `then` que será otra función *callback* que se llamará cuando la petición al servicio o la respuesta fallen.

Más información

Para más información podéis consultar:

- [https://docs.angularjs.org/api/ng/service/\\$q](https://docs.angularjs.org/api/ng/service/$q)
- <http://mcgivery.com/ionic-using-factories-and-web-services-for-dynamic-data/>

Directivas

Según la definición que da Angular: las directivas son marcadores sobre los elementos del DOM (como los atributos, los nombres de elementos o etiquetas, o las clases CSS) que indican al compilador HTML de Angular (`$compile`) que adjunte un determinado comportamiento a dicho elemento del DOM o incluso que lo transforme por un bloque completo de contenido.

Por clarificarlo aún más, las directivas permiten transformar el aspecto y comportamiento del código HTML por otro que definamos nosotros. En Ionic, todas las etiquetas que empiezan con `ion-` son directivas de Angular que tienen un comportamiento y un aspecto asociado. Por ejemplo, la etiqueta:

```
<ion-list>
```

Esta etiqueta se procesa mediante una directiva de Angular y genera el código y comportamiento correspondiente a un listado. Y para que esto funcione, en el código de Ionic hay una directiva que está configurada para gestionar cualquier elemento con el nombre `ion-list` . El código sería el siguiente:

```
IonicModule.directive('ionList', [ '$timeout',  
    function($timeout) {  
        return {  
            restrict: 'E',  
            require: ['ionList', '^?$ionicScroll'],  
            controller: '$ionicList',  
            compile: function($element, $attr) {  
                //... etc ...  
            }  
        }  
    }  
]);
```

En este código hay muchos conceptos nuevos, pero de momento solo nos interesa centrarnos en algunos aspectos clave. La línea `restrict: 'E'` indica si la directiva se refiere a un atributo, a un elemento o una clase CSS. La propiedad `restrict` puede tener los siguientes valores:

- `'A'` – La directiva se refiere solo a nombres de atributos.
- `'E'` – La directiva se refiere solo a elementos (los nombres de las etiquetas).
- `'C'` – Se refiere al nombre de una clase CSS.

- `'AEC'` – Estas letras se pueden usar de forma combinada para por ejemplo referirnos a la vez a atributos, elementos y clases.

En el ejemplo el nombre de la directiva indica `ionList`, pero Angular lo transformará y procesará tanto el nombre `ionList` como `ion-list`. Otro ejemplo, si creamos una directiva para `ionTab` se tendrá en cuenta tanto `ionTab` como los elementos con nombre `ion-tab`.

Ejemplo completo

Como ejemplo vamos a crear una aplicación con dos pantallas. En primer lugar definimos el contenido del fichero `index.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1,
      user-scalable=no, width=device-width">
    <title></title>
    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-app="starter">

    <ion-nav-bar class="bar-positive">
      <ion-nav-back-button class="button-clear">
        <i class="ion-chevron-left padding-left"></i>
      </ion-nav-back-button>
    </ion-nav-bar>
    <ion-nav-view></ion-nav-view>

  </body>
</html>
```

Este fichero carga todas las dependencias e inicia la carga del módulo `starter` de Angular. En el fichero `js/app.js` creamos dicho módulo y definimos ya las rutas que va a tener:

```
angular.module('starter', ['ionic'])

.config(function($stateProvider, $urlRouterProvider) {

  $stateProvider
    .state('home', {
      url: '',
      templateUrl: 'templates/home.html',
      controller: 'homeCtrl'
    })
    .state('about', {
      url: '/about',
      templateUrl: 'templates/about.html',
      controller: 'aboutCtrl'
    });

  // Página por defecto
  $urlRouterProvider.otherwise('');
})
```

Dentro de la carpeta `templates` vamos a colocar las dos plantillas que necesitaremos. La plantilla `home.html` con el siguiente contenido:

```
<ion-view title="Inicio">
  <ion-content>
    Aplicación de ejemplo.

    <a ui-sref="about">Información sobre el autor</a>
  </ion-content>
</ion-view>
```

Y la plantilla `about.html` con el contenido:

```
<ion-view title="Autor">
  <ion-content>
    El autor de esta aplicación es {{author.name}}.
  </ion-content>
</ion-view>
```

A continuación vamos a definir los controladores, los cuales los añadiremos también dentro del fichero `js/app.js`. El controlador para el estado `home` lo dejamos vacío ya que no necesita realizar ninguna acción:

```
.controller('homeCtrl', function($scope) {
  ;
})
```

En el controlador para el estado `about` cargamos el servicio `aboutService` y asignamos el valor devuelto por el método `getAuthor()` a una variable del `$scope` que será sustituida en la vista correspondiente:

```
.controller('aboutCtrl', function($scope, aboutService) {  
    $scope.author = aboutService.getAuthor();  
})
```

Por último añadimos el servicio `aboutService` también al fichero `js/app.js` :

```
.factory('aboutService', function() {  
    return {  
        getAuthor: function() {  
            var author = { name: "Juan" };  
            return author;  
        },  
    }  
})
```

Una vez completado el código podemos probar la aplicación ejecutando en un terminal el comando:

```
$ionic serve
```

Plugins

Instalar *ngCordova*

Para instalar *ngCordova* simplemente tenemos que hacer:

```
$ bower install ngCordova
```

Si nos diese algún error de permisos tendríamos que ejecutar:

```
$ sudo bower install ngCordova --allow-root
```

A continuación tenemos que incluir " `ng-cordova.js` " o " `ng-cordova.min.js` " en el fichero `index.html` justo antes de `cordova.js` y después de incluir AngularJS e Ionic (ya que *ngCordova* depende de AngularJS):

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
<script src="cordova.js"></script>
```

Para usarlo tenemos que inyectarlo en el módulo como una dependencia de Angular, por ejemplo:

```
angular.module('myApp', ['ngCordova'])
```

Uso

Añadir un plugin:

```
$ ionic plugin add https://github.com/EddyVerbruggen/Toast-PhoneGap-Plugin.git
```

Ver la lista de plugins instalados:

```
$ cordova plugin list
```


Para eliminar usamos `plugin remove` con el nombre del plugin a eliminar. Este nombre puede no ser el mismo que el que usamos para instalar, así que lo tenemos que consultar usando: `cordova plugin list`. Y por último lo eliminamos de la forma:

```
$ ionic plugin remove nl.x-services.plugins.toast
```

A veces después de instalar un plugin (como el de sqlite) no se instala todo lo necesario en la plataforma. Por lo que se recomienda hacer:

```
$ ionic platform rm [ios/android]
$ ionic platform add [ios/android]
```

Para más información podéis consultar:

<http://ngcordova.com/docs/plugins/>

Whitelist

Si aparece el error:

```
No Content-Security-Policy meta tag found.
Please add one when using the cordova-plugin-whitelist plugin.
```

Hay que instalar el siguiente plugin

```
$ ionic plugin add cordova-plugin-whitelist
```

Si se van a hacer peticiones de contenido a un servidor externo hay que definir la política de seguridad añadiendo a la cabecera de la página la siguiente directiva:

```
<meta http-equiv="Content-Security-Policy" content="default-src *; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-inline' 'unsafe-eval'">
```

Para más información podéis consultar:

<https://github.com/apache/cordova-plugin-whitelist#content-security-policy>

Toast

Por ejemplo, para usar el *plugin* para mostrar un *toast* tenemos que instalar:

```
$ ionic plugin add https://github.com/EddyVerbruggen/Toast-PhoneGap-Plugin.git
```

Más información en: <http://ngcordova.com/docs/plugins/toast/>

SQLite

Por ejemplo para usar el *plugin* para trabajar con *SQLite* instalamos:

```
$ ionic plugin add https://github.com/litehelpers/Cordova-sqlite-storage.git
```

A continuación se incluye un ejemplo de uso desde un controlador:

```
module.controller('MyCtrl', function($scope, $cordovaSQLite) {

    var db = $cordovaSQLite.openDB({ name: "my.db" });

    // for opening a background db:
    //var db = $cordovaSQLite.openDB({ name: "my.db", bgType: 1 });

    $scope.execute = function() {
        var query = "INSERT INTO test_table (data, data_num) VALUES (?,?)";
        $cordovaSQLite.execute(db, query, ["test", 100]).then(function(res) {
            console.log("insertId: " + res.insertId);
        }, function (err) {
            console.error(err);
        });
    };
});
```

Más información en: <http://ngcordova.com/docs/plugins/sqlite/>

Errores comunes

Todas las llamadas a *plugins* o eventos de Cordova tienen que esperar a que se cargue la librería. Para esto en primer lugar hay que escuchar el evento `deviceready` de la forma:

```
document.addEventListener("deviceready", function ()
{
    // Ya podemos usarlo...
    $cordovaPlugin.someFunction().then(success, error);
}, false);

// O también podemos esperar al evento usando el código:

$ionicPlatform.ready(function() {
    // Ya podemos usarlo...
    $cordovaPlugin.someFunction().then(success, error);
});
```

Publicación

Una vez terminado el desarrollo de una aplicación el último paso es su publicación en los *market* de aplicaciones. Para esto lo primero que tenemos que hacer es generar la versión *release* (o versión final, sin código de depuración) de cada una de las plataformas en la que la queramos publicar.

En primer lugar, antes de compilar la versión final, tendremos que revisar los *plugins* que hemos utilizado durante el desarrollo, ya que si hemos añadido alguno de depuración o *profiling* que pueda ralentizar la ejecución lo deberíamos de quitar para la versión de producción. Por ejemplo, por defecto viene incluido el *plugin* para la mostrar el texto de depuración por consola, pero este módulo no es necesario para la versión final y lo deberíamos quitar:

```
$ cordova plugin rm cordova-plugin-console
```

A continuación, para generar la versión *release* para Android (o para la plataforma que queramos) tendríamos que escribir el siguiente comando:

```
$ cordova build --release android
```

Este comando compilará la aplicación basándose en la configuración indicada en el fichero `config.xml` y colocando el fichero generado en la carpeta `build` correspondiente. Por ejemplo para Android lo podremos encontrar en `platforms/android/build/outputs/apk`. Lo único que faltaría sería firmar la aplicación para poder publicarla.

Para más información sobre el proceso de publicación y notas específicas sobre la versión *release* de cada plataforma podéis consultar la siguiente documentación:

<http://ionicframework.com/docs/guide/publishing.html>

Generar iconos y *splashscreen*

A la hora de publicar una aplicación también tendremos que actualizar los iconos de la misma y la imagen de *splashscreen* en caso de que la hayamos utilizado. Para esto Ionic incluye una serie de herramientas que nos facilitará mucho el trabajo. Podéis encontrar un tutorial de como hacerlo en la siguiente dirección:

<http://blog.ionic.io/automating-icons-and-splash-screens/>

Más información

Documentación

- Primeros pasos: <http://ionicframework.com/getting-started/>
- Documentación: <http://ionicframework.com/docs/>
- Componentes: <http://ionicframework.com/docs/components/>
- API de Ionic: <http://ionicframework.com/docs/api/>
- Más documentación como vídeos, trucos, foro, etc. en: <http://learn.ionicframework.com/>

Recursos para Ionic

- Iconos: <http://ionicons.com/>
- Código: <http://codecanyon.net/category/mobile/native-web>
- Previsualización de las Apps: <http://view.ionic.io>
- ngCordova - Versión de Cordova para Angular: <http://ngcordova.com>
- Ionic creator - Crea pantallas básicas de Ionic visualmente: <http://creator.ionic.io>
- Ejemplos de apps hechas con Ionic: <http://showcase.ionicframework.com/>
- Libro sobre Ionic: http://manning.com/wilken/?a_aid=ionicinactionben&a_bid=1f0a0e1d

Consejos y solución de errores

- A year using Ionic to build hybrid applications: <http://www.airpair.com/javascript/posts/a-year-using-ionic-to-build-hybrid-applications>
- What I learned building an app with Ionic Framework: <http://www.betsmartmedia.com/what-i-learned-building-an-app-with-ionic-framework>
- 5 Ionic Framework App Development Tips and Tricks: <http://www.sitepoint.com/5-ionic-app-development-tips-tricks/>
- Structure of an Ionic App: <http://mcgivery.com/structure-of-an-ionic-app/>
- <https://jsjutsu.com/2015/06/21/tutorial-ionic-framework-parte-1/>

Plugins de Angular interesantes

- Plugin con multitud de filtros para Angular: <https://github.com/a8m/angular-filter>

Ejercicios 1

En esta sección de ejercicios vamos a desarrollar la primera parte de una aplicación destinada a la gestión de una biblioteca de un centro educativo.

Ejercicio 1.1 - Inicio de la aplicación (1 punto)

En primer lugar crea un nuevo proyecto mediante el *cli* de Ionic, usa la plantilla en blanco (tipo *blank*) y llama al proyecto "iBiblioteca". Al crear un proyecto de este tipo se incluirá el contenido mínimo de ejemplo dentro de la carpeta `www`, pero es suficiente para empezar nuestro proyecto.

Partiendo del contenido del fichero `index.html` crea una pantalla similar a la de la siguiente captura:



Instrucciones:

- En la parte superior añade un *header* con el título "Inicio".
- Añade la clase CSS *padding* a la sección de contenido para dejar un pequeño margen de borde.

- Dentro del área de contenido crea dos tarjetas usando el elemento *cards* con el contenido indicado.
- En el botón "Acceso a iBiblioteca" crea un enlace a "ibiblioteca.html".
- En el botón "Consultar información" crea un enlace a "author.html".

Ejercicio 1.2 - Página del autor (1 punto)

En este ejercicio vamos a crear un nuevo documento html llamado "author.html" con información sobre el autor. Esta página contendrá una cabecera con el título "Autor" y un botón atrás con un enlace a "index.html". En la sección principal de contenido le añadiremos la clase css *padding* para dejar un poco de margen de borde e insertaremos una tarjeta usando el elemento tipo *card* con el siguiente contenido:



Donde:

- El botón "Currículum Vitae" apuntará a otra página llamada "curriculum.html".
- El botón de correo será un enlace tipo "mailto:gmail@gmail.com".
- El botón de Twitter será un enlace normal a una cuenta de Twitter.
- El botón de teléfono será un enlace del tipo "tel:900900900".

A continuación tenemos que crear otra página html llamada "curriculum.html". Esta página tendrá una cabecera con el título "Currículum" y un botón atrás que nos llevará a la página "author.html". En la sección de contenido insertaremos un elemento tipo lista con los

estudios realizados o los últimos puestos ocupados. Por ejemplo:



Nota: al usar un elemento tipo lista no tendréis que poner *padding* a la sección de contenido.

Ejercicio 1.3 - Listado de libros (1 punto)

En este ejercicio vamos a crear un documento html llamado "ibiblioteca.html" en el que mostraremos un listado de los libros de nuestra biblioteca. Esta página tendrá en primer lugar una cabecera con el título "iBiblioteca" y un botón atrás que nos llevará al "index.html". En la sección de contenido añadiremos un elemento tipo lista para mostrar los libros con un formato similar al de la siguiente imagen:

 iBiblioteca



La colmena
Autor: Camilo José Cela Trulock
Año: 1951



La galatea
Autor: Miguel de Cervantes Saave...
Año: 1585



El ingenioso hidalgo don Quijot...
Autor: Miguel de Cervantes Saave...
Año: 1605



La dorotea
Autor: Félix Lope de Vega y Carpio
Año: 1632



La dragontea
Autor: Félix Lope de Vega y Carpio
Año: 1602

En la plantilla de estos ejercicios tenéis disponible un XML con los datos de los libros de ejemplo y sus correspondientes imágenes para las portadas.

Esta pantalla de momento no tiene que realizar ninguna acción más, solamente mostrar el listado de libros. En los ejercicios de las siguientes secciones lo completaremos con mas funcionalidad.

Ejercicios 2

En esta segunda sección de ejercicios vamos a continuar con el ejercicio de la biblioteca para añadirle la arquitectura de Angular y algunos componentes con contenido dinámico.

Ejercicio 2.1 - Configuración, rutas y controladores (1 punto)

Vamos a modificar el fichero `js/app.js` para añadir las cuatro rutas de nuestra aplicación y los correspondientes controladores. Este archivo ya tendrá algo de código que viene incluido al crear la aplicación con Ionic, simplemente se crea el módulo principal (comprobar que se llame `starter`) y se inicializan algunos componentes necesarios para que funcione Ionic (dentro de la sección `.run`).

A continuación de este código vamos a añadir la configuración de la aplicación, en la siguiente tabla se muestra un resumen con todos los campos necesarios para que la creéis:

<i>state</i>	<i>url</i>	<i>templateUrl</i>	<i>controller</i>
home	"	home.html	homeCtrl
author	/author	author.html	authorCtrl
curriculum	/curriculum	curriculum.html	curriculumCtrl
ibiblioteca	/ibiblioteca	ibiblioteca.html	ibibliotecaCtrl

En este mismo fichero añadiremos los cuatro controladores. El controlador `homeCtrl` estará vacío. El controlador `authorCtrl` solo asignará al `$scope` un objeto como el siguiente (pero rellenado con los datos del autor correspondiente):

```
$scope.author = {  
  name: '',  
  email: '',  
  twitter: '',  
  phone: ''  
};
```

El controlador `curriculumCtrl` también asignará directamente al `$scope` un array con la lista de puestos ocupados. Esta lista la podéis rellenar con el contenido que ya teníais para esta pantalla:

```
$scope.works = [
  { date: '', description: '' },
  { date: '', description: '' },
  { date: '', description: '' }
];
```

Y por último, el controlador `ibibliotecaCtrl` asignará al `$scope` un array con la lista de libros. Para obtener esta lista usaremos el servicio `bookService`, así que en primer lugar pegaremos el siguiente código al final de nuestro fichero `js/app.js` para definir dicho servicio:

```
.factory('bookService', function() {
  var books = [
    { title: 'La colmena', year: '1951', author: 'Camilo José Cela Trulock',
      isbn: '843992688X', editorial: 'Anaya', cover: 'lacolmena.jpg' },
    { title: 'La galatea', year: '1585', author: 'Miguel de Cervantes Saavedra',
      isbn: '0936388110', editorial: 'Anaya', cover: 'lagalatea.jpg' },
    { title: 'El ingenioso hidalgo don Quijote de la Mancha', year: '1605',
      author: 'Miguel de Cervantes Saavedra',
      isbn: '0844273619', editorial: 'Anaya', cover: 'donquijote.jpg' },
    { title: 'La dorotea', year: '1632', author: 'Félix Lope de Vega y Carpio',
      isbn: '847039360X', editorial: 'Anaya', cover: 'ladorotea.jpg' },
    { title: 'La dragontea', year: '1602', author: 'Félix Lope de Vega y Carpio',
      isbn: '8437624045', editorial: 'Anaya', cover: 'ladragontea.jpg' }
  ];
  return {
    getBooks: function() {
      return books;
    },
    getBook: function(id) {
      return books[id];
    },
  }
});
```

Y para utilizarlo tenemos que inyectar el servicio en el nuevo controlador `ibibliotecaCtrl` y simplemente asignar los libros al `$scope` de la forma:

```
$scope.books = bookService.getBooks();
```

Ejercicio 2.2 - Página de inicio y *templates* (1 punto)

En este ejercicio vamos a actualizar las vistas para que todo funcione correctamente. Cada una de las páginas que habíamos creado en el ejercicio anterior (`author.html` , `curriculum.html` e `ibiblioteca.html`) las movemos dentro de la carpeta `templates` . Además añadimos la plantilla `home.html` que de momento estará vacía. De esta forma en la carpeta raíz nos habrá quedado solo el fichero `index.html` .

A continuación vamos a arreglar el contenido de las plantillas. Dentro de cada vista tendremos que dejar una estructura similar a la siguiente:

```
<ion-view title="Inicio">
  <ion-content class="padding">
    ...CONTENIDO ANTERIOR...
  </ion-content>
</ion-view>
```

El resto del contenido de la vista que teníamos lo eliminamos, quitamos todas las etiquetas HTML iniciales para la carga de dependencias, etc., y dejamos solamente el bloque de contenido marcado con `<ion-content>` . Las cabeceras también tendremos que eliminarlas, pero moveremos el título al atributo `title` de la directiva `ion-view` de cada vista. Para el template `home.html` moveremos el contenido que teníamos en `index.html` siguiendo estas mismas instrucciones. Y en `index.html` cambiaremos su sección `<body>` para que incluya únicamente las siguientes directivas:

```
<body ng-app="starter">
  <ion-nav-bar class="bar-positive">
    <ion-nav-back-button class="button-clear">
      <i class="ion-chevron-left padding-left"></i>
    </ion-nav-back-button>
  </ion-nav-bar>
  <ion-nav-view></ion-nav-view>
</body>
```

A continuación vamos a arreglar los enlaces que cambiaban de pantalla, en `home` teníamos dos y en `author` teníamos otro para ir a `curriculum` . Tenemos que eliminar el atributo `href` de estos enlaces y sustituirlo por `ui-sref` indicando el nombre del estado al que queremos ir, por ejemplo:

```
<a ui-sref="author" class="item item-icon-left">...</a>
```

Por último vamos a modificar las plantillas para que muestren el contenido que le hemos pasado desde el controlador a través de la variable `$scope` . En `author.html` mostraremos los valores directamente usando, por ejemplo, `{{author.name}}` . Para `curriculum.html` e

`ibiblioteca.html` tendremos que crear un bucle que muestre todos los elementos del array y en cada uno de ellos genere una fila de la lista, por ejemplo:

```
<li class="item" ng-repeat="work in works">
  <p class="date">{{work.date}}:</p>
  <p>{{work.description}}</p>
</li>
```

Ejercicio 2.3 - Pantalla vista detalle (1 punto)

Por último vamos a añadir una pantalla para mostrar la vista detalle de un libro de la biblioteca. Es decir, que al pulsar sobre un elemento de la lista se cambie a una nueva pantalla en la que se muestre la información ampliada de un libro.

Vamos a empezar definiendo la nueva ruta, la cual tendrá la siguiente configuración:

<i>state</i>	<i>url</i>	<i>templateUrl</i>	<i>controller</i>
bookDetail	/book/:id	book.html	bookCtrl

A continuación vamos a crear el controlador `bookCtrl`, el cual tendrá que obtener el valor de `id` de los parámetros de entrada (revisa la sección sobre Controladores de la teoría). A partir del valor de `id` podemos recuperar los datos del libro usando el servicio `bookService` y asignarlo al `$scope` de la forma:

```
$scope.book = bookService.getBook(id);
```

Por último tenemos que terminar las vistas para que todo funcione. En la vista que ya teníamos `ibiblioteca.html` vamos a modificar los elementos del listado para añadir un enlace a la pantalla detalle:

```
<div class="list">
  <a class="item item-thumbnail-left" ng-repeat="book in books"
    ui-sref="bookDetail({id: $index})">
    ...
  </a>
</div>
```

Como se puede ver el enlace al estado `bookDetail` se ha creado usando el atributo `ui-sref` y pasándole como parámetro la variable `$index`. Esta variable contiene la posición del elemento actual en el array iterado.

En la nueva plantilla `book.html` mostraremos la información ampliada de un libro. A continuación se muestra una captura de como tendrá que quedar esta pantalla:

< iBiblioteca La colmena



La colmena

Autor: Camilo José Cela Trulock

Año: 1951

ISBN: 843992688X

Editorial: Anaya