Práctica de AngularJS + HTML5 + CSS3

Jeviteca

Jeviteca es una aplicación web que lista los mejores álbumes y bandas de metal de todos los tiempos. Podemos consultar la información por banda, álbum y género. Todos los recursos necesarios -metadatos e imágenes- podéis descargarlos desde el mismo post donde descargaste este documento.

¿Qué tengo que hacer?

Tienes que programar una aplicación web que ponga en práctica todo lo visto durante el curso de AngularJS + HTML5 + CSS3. No vas a necesitar nada más que eso, por tanto evita complicarte la vida intentando programar alguna funcionalidad con jQuery, por ejemplo. Si quieres que la aplicación quede algo más *atractiva*, siéntete libre de usar algún framework responsive CCS3 tipo Bootstrap o similar, pero no es obligatorio en ningún momento.

¿Qué debe tener la aplicación? Pues sí o sí, debe tener 3 secciones o pestañas que muestren la siguiente información:

- La pestaña `Álbumes` muestra una colección de álbumes.
- La pestaña `Bandas` muestra una colección de bandas.
- La pestaña `Géneros` muestra una colección de géneros.

En cada una de estas pestañas se pintará una tabla con los datos que se consideren útiles para cada caso.

¿Y esto es todo? Sí, esto es todo. Una web con tres pestañas que muestran tablas con información de álbumes, bandas y géneros de música. **Pero no corras tanto, que hay algunas condiciones.**

- Tienes que familiarizarte con los **Filtros** de AngularJS, por tanto haz uso de ellos en aquellos sitios que los pidan a gritos.
- Cada elemento de cada tabla es una **Directiva** de AngularJS. Tendrás que crear tres, una para cada tipo de elemento -álbum, banda y género-.
- Gran parte de la información que se muestra es de solo lectura. Es responsabilidad tuya, como programador, crear el código lo más óptimo posible: recuerda usar **One-time binding** cuando se pueda.
- Los settings de la aplicación deben establecerse donde el señor AngularJS nos recomienda, y que no es otro lugar que en un **Value** o en un **Constant**.
- El usuario debe poder marcar sus álbumes, bandas y géneros favoritos de manera que pueda

consultarlos cada vez que use la aplicación. A falta de una cuenta de usuario *per se*, una buena solución puede ser hacer uso del **Local Storage** del navegador.

Ya no parece tan sencillo, ¿verdad? No te preocupes, te explicamos todo poco a poco.

Modelo

El modelo de la aplicación se compone de varias entidades. Son las descritas a continuación.

Entidad `album`:

- id `int`
- title `string`
- year `int`
- band `band object`
- genre `genre object`
- image `string`
- tracklist `string array`

Entidad `band`:

- id `int`
- name `string`
- origin `string`
- members `member object array`

Entidad `genre`:

- id `int`
- name `string`

Entidad `member`:

- name `string`
- instruments `string array`

Sirva el siguiente objeto JSON como ejemplo de una entidad `album` cualquiera:

```
{
    "id": 6,
    "title": "Once More 'Round the Sun",
    "year": 2014,
    "band": {
        "id": 1,
        "name": "Mastodon",
        "origin": "Atlanta, Georgia, Estados Unidos",
        "members": [
            {
                "name": "Troy Sanders",
                "instruments": ["Voz", "Bajo"]
            },
            {
                "name": "Brent Hinds",
                "instruments": ["Voz", "Guitarra"]
            },
            {
                "name": "Bill Kelliher",
                "instruments": ["Guitarra"]
            },
            {
                "name": "Brann Dailor",
                "instruments": ["Voz", "Batería"]
            }
        ]
    },
    "genre": {
        "id": 4,
        "name": "Sludge metal"
    },
    "image": "6-mastodon-once-more-round-the-sun.jpg",
    "tracklist": [
        "Tread Lightly",
        "The Motherload",
        "High Road",
        "Once More 'Round the Sun",
        "Chimes at Midnight",
        "Asleep in the Deep",
        "Feast Your Eyes",
        "Aunt Lisa",
        "Ember City",
        "Halloween",
        "Diamond in the Witch House"
    ]
}
```

Con el fin de facilitar la obtención de la información en las distintas secciones de la apliación, la colección de álbumes viene dada en tres sabores:

- El documento `albums.json` contiene la colección dispuesta por álbumes.
- El documento `bands.json` contiene la colección dispuesta por bandas.
- El documento `genres.json` contiene la colección dispuesta por géneros.

Single Page Application

Tienes que tener en cuenta que la barra de navegación de la aplicación debe estar siempre visible, sea cual sea la sección por la que se navegue. Vas a necesitar para ello el módulo angular-route-segment. Recuerda que con este módulo lo que hacemos es definir zonas del DOM que son dinámicas, y por tanto susceptibles de verse modificadas. La barra de navegación debe pintarse en el `index.html` y debajo de ella un segmento dinámico; de esta forma, podrás utilizar el contenido de dicho segmento para pintar la información de las distintas secciones.

¿Recuerdas dónde se establecen segmentos y las rutas de navegación con sus correspondientes controladores y vistas? Efectivamente, en el documento `app.js` durante la fase config de la aplicación. Ya de paso, utiliza esa genialidad llamada `resolve` para inyectar como dependencia a cada controlador los datos que va a necesitar. Era algo parecido a esto:

```
angular.module("jeviteca").config(function($routeSegmentProvider) {
    $routeSegmentProvider.when("/bands", "bands");
    $routeSegmentProvider.segment("bands", {
        controller: "BandsController",
        templateUrl: "views/Bands.html",
        resolve: {
            Bands: ["BandsProvider", function(BandsProvider) {
                return BandsProvider.getBands();
            }]
        }
    });
});
```

Servicios

Recuerda que los **servicios** de AngularJS son objetos singleton y por ello son los más indicados para ocuparse de tareas comunes de la aplicación. Generalmente, el acceso a datos lo haremos desde servicios. Un código similar al siguiente puede servirte:

```
angular.module("jeviteca").service("BandsProvider", function($http) {
    this.getBands = function() {
        return $http.get("data/bands.json");
    };
});
```

Debes crear un servicio para cada colección de álbumes:

- Servicio `AlbumsProvider`, que trabaja con `albums.json`.
- Servicio `BandsProvider`, que trabaja con `bands.json`.
- Servicio `GenresProvider`, que trabaja con `genres.json`.

Es posible que para alguna funcionalidad concreta necesites obtener una entidad individual, no la colección completa. El servicio `$filter` puede solucionarte el problema:

```
function getBandById(bandId) {
    return $filter("filter")(bandsCollection, {"id": bandId})[0];
}
```

Filtros

Los filtros, al retornar una función en vez de un objeto, se antojan algo liosos para trabajar con ellos, pero la verdad es que son muy útiles. Solo aplican formato a los datos de entrada, no los modifican.

Dejo a tu criterio cuáles crear, pero lo ideal es que programes al menos un par de ellos de tu cosecha. Por ejemplo, y este como te lo doy queda descartado, podrías presentar la lista de instrumentos de los miembros de las bandas como una cadena de texto separada por comas:

```
angular.module("jeviteca").filter("instruments", function() {
    return function(collection) {
        return "Instrumentos: " + collection.join(", ");
    };
});
```

Directivas

Como te decía más arriba, los elementos que se muestran en cada tabla deberían ser directivas. Pinta en cada una de ellas los datos que más relevantes consideres. A mí me puede parecer interesante algo de este estilo:

```
// Vista
<tr>
    <td>{{ band.name }}</td>
    <td>{{ band.origin }}</td>
    <td>
        <ul>
            <li ng-repeat="member in band.member">{{ member.name }}</li>
        </ul>
    </td>
</tr>

// Directiva
angular.module("jeviteca").directive("band", function() {
    return {
        templateUrl: "band.html",
        scope: {
            band: "="
        }
    };
});
```

Favoritos

Como decía más arriba, una buena forma de implementar un sistema de favoritos sin tener una cuenta de usuario es a través de `Web Storage`. Cuando vimos HTML5 ya comentamos esta API, cuyo cometido no es otro que almacenar datos de manera local en el navegador del usuario.

En cada elemento listado en una tabla, debes facilitar un enlace o icono que establezca dicho elemento como favorito.

Recuerda un par de detalles de la API:

- Tienes que comprobar siempre si el navegador puede hacer uso de la misma.

```
if (typeof(Storage) !== "undefined") {
    // Código
}
```

- El almacén solo soporta `string`, por tanto tendrás que serializar y deserializar los datos al guardar y obtener.

```
// Por ejemplo, para almacenar...
localStorage.setItem("favBands", JSON.stringify(data));

// ...y obtener JSON
JSON.parse(localStorage.getItem("favBands"));
```

Keep coding

Todo lo comentado hasta este punto es la práctica obligatoria. A partir de aquí no es obligatorio, pero sí extremadamente recomendable hacer. Por ahora has puesto en práctica gran parte de los conceptos vistos durante todo el curso, pero es el momento de apretar un poco la maquinaria y empezar a *andar sin ruedines*. Voy a proponerte una serie de cosas de la aplicación que creo que son mejorables, a ver qué te parecen. Como te he dicho, no es obligatorio pero sí que deberías seguir jugando con el desarrollo web, sobre todo con AngularJS.

- Me parece una falta de respeto al usuario *jevi* medio que la aplicación no tenga una vista en detalle de la banda o del álbum. ¿Tienes las tablas de bandas o álbumes muy cargadas de datos? Pues no es la idea. Estas tablas deberían ser algo más livianas y delegar a una vista en detalle el resto de datos. Por tanto, crea una vista en detalle de bandas y otra de álbumes, de manera que desde las tablas correspondientes se pueda *clicar* sobre los elementos y navegar a su información extendida.

- Una vez hecho el punto anterior, te habrás dado cuenta de que tampoco tienes muchos detalles que mostrar, sobre todo de las bandas. ¿Sabes qué puede ser interesante? Que en la vista extendida de bandas pongas un enlace a la Wikipedia que lleve al artículo de la banda en cuestión. No vayas a buscar en Wikipedia banda por banda para obtener el enlace; somos más listos y más vagos que eso. Usa lo siguiente:

```
function getWikipediaLink(bandName) {
    var query = encodeURIComponent(bandName);
    return "https://es.wikipedia.org/wiki/Special:Search?search=" + query;
}
```

- Y para la vista extendida de los álbumes sería genial poder ir a YouTube y ver los vídeoclips de los distintos temas. Te ayudo un poco:

```
function getYouTubeLink(bandName, trackName) {
    var query = encodeURIComponent((bandName + " " + trackName).toLowerCase());
    return "https://www.youtube.com/results?search_query=" + query;
}
```

- Las tablas que muestran las bandas, álbumes y géneros deberían poder filtrarse. Si muestran muchos

elementos puede ser incómodo navegar por ellos para localizar lo que se está buscando. Por tanto, pon algún `<input>` donde poder escribir para filtrar la tabla en tiempo real por uno o varios criterios.

- Es de vital interés para el señor *jevi* poder valorar cada álbum. ¿No sería genial poner en la vista de detalles de álbumes un sistema de valoración? Una vez el usuario valore de 0 a 5 un álbum, deberá almacenarse de alguna forma para que perdure este dato en el tiempo. Las siguientes veces que el usuario navegue ese álbum verá su valoración de manera gráfica con una colección de estrellas rellenas y estrellas vacías -por poner- que la represente. Si me tiras de la lengua un poco te diré que yo crearía una directiva nueva para todo este tinglado. Si el álbum no está valorado, muestra unos botones `radio` o un `combo` con los posibles valores. Cuando se valora, el valor seleccionado se almacena en `Web Storage` de manera que sea recuperable a futuro. Por último, en caso de existir valoración, se recupera y se pintan tantas estrellas rellenas y estrellas vacías como corresponda. Pero esto es solo lo que yo haría pensando en practicar directivas con vistas dinámicas, que es una de tantas formas de hacerlo; si a ti te mola más otra forma de llevarlo a cabo, siempre que respete la metodología de trabajo con AngularJS, adelante con ella.

Keep in mind

- Intenta usar siempre la anotación de array en línea para la inyección de dependencias. Es fea, pero también la recomendada.
- No olvides referenciar cada documento `.js` que crees en el `index.html`, que luego no sabes de dónde vienen los problemas.
- Evita llamar a tus directivas `ngX` y a tus servicios `$x`, puesto que es como AngularJS llama a sus propias directivas y servicios.
- La manipulación del DOM siempre va en directivas. ¿Podría funcionar fuera de una directiva? Por poder... ¿Pero fallaría? Sí, en alguna vuelta del *digest loop* se le acaba yendo la pinza, de ahí el invento de las directivas.
- Si ves que tienes código replicado en varios controladores ya sabes lo que tienes que hacer: llevarlo a un servicio e inyectar dicho servicio como dependencia en los controladores para poder usar dicho código.
- Recuerda que los nombres de filtros y directivas son siempre **lowerCamelCase**. Suele olvidarse la parte del *lower* y tardas un buen rato en darte cuenta del problema.
- Una torpeza muy común es usar la URL de una vista en la propiedad `template` en vez de `templateUrl`. Y hasta que caes en el detalle pierdes *siglos*.
- AngularJS carga las vistas HTML con el servicio `$http` (vía Ajax) y por temas de seguridad los navegadores bloquean toda petición de documentos locales. Si te ocurre, revisa que tienes

correctamente levantado el servidor web de estáticos y la petición de la vista apunta a `http://localhost` y no a `file:///`.

- Esto es The Matrix, no lo olvides. Pregunta al oráculo cada duda que tengas, menos cómo doblar cucharas con la mente, que no lo tengo dominado aún.