INGENIERÍA EN SOFTWARE

Aplicaciones Web

Practica – 2

OBJETIVOS DEL LABORATORIO

- Comprender la estructura de un proyecto JavaScript Backbone.JS
- Manipular elementos HTML con JavaScript (DOM básico).
- Trabajar con varias vistas de manera coordinada y algo más profesional.

ACTIVIDADES DEL LABORATORIO

MINIPROYECTO DE APLICACIÓN CON BACKBONE Y MARIONETTE

En esta sesión vamos a desarrollar una pequeña aplicación algo más compleja que los ejercicios de las otras sesiones. El objetivo es trabajar con varias vistas de manera coordinada y algo más realista que lo que hemos hecho hasta ahora.

5.1. REQUERIMIENTOS

Queremos desarrollar una pequeña aplicación de gestión de comics en la que podamos buscar comics de Marvel usando su API REST. Si estamos autentificados también podremos marcar comics como favoritos, y gestionar luego esta lista de favoritos.

Como requerimientos "iniciales":

- Se podrán buscar comics por título, listando los resultados de modo resumido
- Se podrán ver todos los detalles de un comic determinado

Como requerimientos "adicionales"

Se usará Kinvey, FireBase, Supabase, Backendless como ***backend*** para realizar las siguientes operaciones:

- Se podrá hacer ***login*** y ***logout***
- Si estamos logueados, se podrán marcar comics como favoritos, añadiéndolos a una lista de "mis comics".
- Se podrán eliminar comics de la lista "mis comics".

Fíjate que la aplicación trabaja con dos ***backends***. El servidor de Marvel solo nos permite leer datos. En él buscaremos comics y obtendremos sus datos. El servidor de Kinvey nos permite también guardar información, así que ahí es donde guardaremos los datos de nuestros comics favoritos.

Daremos aquí instrucciones paso a paso de cómo implementar los requerimientos "iniciales", pero solo algunas guías genéricas de cómo implementar los "adicionales".

Para hacer peticiones al API de Marvel hay que tener un "API key". En la plantilla del proyecto se usa la API Key del profesor (tiene unas 3000 llamadas diarias de límite, seguramente más que suficientes). No obstante, si deseas usar tu propia API key, puedes darte de alta como desarrollador en la web de Marvel para obtenerla, accediendo luego a tu panel de control de desarrollador.

5.2. Implementación de los requerimientos "iniciales"

Aquí tienes la plantilla de proyecto (en la tarea se adjunta la plantilla) que puedes usar como base para tu código.

5.2.1. La "lógica de negocio"

para que funcionen las llamadas al API de Marvel con la clave del profesor, la página debe ser servida por un servidor web que funcione en localhost (si abrís el .html dando doble clic y probáis una llamada al API dará error HTTP 409). Podéis poner en marcha un servidor web simple yendo al directorio donde está la aplicación y ejecutando python -m SimpleHTTPServer, que empezará a funcionar por http://localhost:8000. También podéis instalar un servidor basado en Node con sudo npm install -g http-server y luego ejecutarlo con http-server.

La plantilla usa espacios de nombres y módulos al estilo "patrón módulo". Tienes ya implementado un modelo (Comic) y una colección (Comics). Esta última ya implementa la comunicación con el API de Marvel, para buscar comics por título. Puedes ir a la consola Javascript y teclear:

```
lista = new Marvel.Collections.Comics()
lista.buscar("Hulk")
```

Pasados unos segundos si examinas la variable "lista" debería contener un conjunto de modelos Comic cada uno con los datos de un comic. Aunque el API está paginado y podríamos irle solicitando resultados de 20 en 20 como mucho, para simplificar la aplicación nos "conformaremos" con listar solo los 20 primeros (podría haber menos), no es necesario implementar paginado.

Vamos a usar Marionette para la interfaz porque simplifica bastante el renderizado y la gestión dinámica de las vistas, pero también se podría implementar con Backbone "puro". Puedes hacerlo así si eres lo suficientemente masoquista.

5.2.2. Estructura de la interfaz

Dividiremos la pantalla en tres secciones, representadas con tres div en el index.html

- Parte superior ("cabecera"): aquí aparecerá el formulario de login, y si ya estamos autentificados, un botón o enlace para gestionar "mis comics".

- Parte media ("formBusqueda"): el formulario de búsqueda
- Parte inferior ("listado"): Esta parte irá cambiando. Si hemos buscado mostrará la lista de resultados de búsqueda. Si hemos elegido "mis comics" aparecerán los que hemos ido seleccionando. Si pulsamos sobre "ver detalles" de un comic se verá únicamente el cómic seleccionado.

aunque hablemos de parte superior, media e inferior, la colocación en pantalla es libre. Con el CSS adecuado podrías poner por ejemplo la página a tres columnas y colocar en cada columna una sección. Pero sí deberías respetar las tres secciones y el papel que desempeña cada una de ellas. Por otro lado, dale prioridad a la funcionalidad sobre la interfaz, no te preocupes demasiado del aspecto estético, no es el objetivo del ejercicio.

Estas tres secciones estarán controladas por una "vista global" de Marionette, que gestionará a las vistas "hijas". Pero vamos por partes. Primero vamos a ver y probar las vistas básicas, para ver un solo comic y un listado de comics.

5.2.3. VISTA DE UN SOLO COMIC: JS/VIEWS/VISTACOMIC.JS

La clase ya la tienes definida, ya que en Marionette solo hace falta referenciar dónde está la ***template***, el ***render*** lo implementa Marionette. Eso sí, tendrás que definir tú la ***template*** usando Mustache en el script id="VistaComicTmpl", que ahora está vacío. Muestra al menos el title y la description del comic. Además muestra una imagen a tamaño reducido. Las imágenes se obtienen concatenando:

- Una URL (propiedad thumbnail.path)
- Un tipo de imagen (que puede ser "standard_small", "standard_medium", "standard_large",… consulta aquí todos los tipos o ***image variants*** y escoge el que prefieras).
- Una extensión de archivo (normalmente .jpg pero puede variar, así que se usa el atributo thumbnail.extension para ser más genérico).

Para comprobar que funciona, desde la consola de Javascript lanza una búsqueda de comics como has hecho antes, asegúrate de que ya ha respondido el servidor (la colección no está vacía) y luego crea una vista pasándole una posición de la colección y renderízala en la página. Algo como:

```
lista = new Marvel.Collections.Comics()
lista.buscar("Hulk")
...espera unos segundos, asegúrate de que "lista" contiene datos
v = new Marvel.Views.VistaComic({model:lista.at(0)});
v.render().$el.appendTo('body');
```

Deberían aparecer en pantalla los datos del comic.

5.2.4. VISTA DE LISTA DE COMICS

Esta vista es del tipo CollectionView, lo que quiere decir que no tiene HTML propio, su HTML es solo una concatenación de los HTML de las vistas "hijas" (del tipo VistaComic). Por tanto no tiene ***template***.

Si quisieras que esta vista tuviera HTML propio y por tanto ***template*** (por ejemplo para mostrar un título "Lista de resultados") tendrías que cambiar el tipo por CompositeView.

No obstante para asegurarte de que todo es correcto puedes hacer una prueba similar a la que has hecho para VistaComic, pero ahora para mostrar un listado completo

```
lista = new Marvel.Collections.Comics()
lista.buscar("Hulk")
...espera unos segundos, asegúrate de que "lista" contiene datos
v = new Marvel.Views.VistaComics({collection:lista});
v.render().$el.appendTo('body');
```

Ya estamos seguros de que al menos las piezas básicas con los datos de los comics funcionan. Vamos con las piezas de "alto nivel".

5.2.5. VISTA GLOBAL

Tienes el esqueleto en js/views/VistaGlobal.js. Es una vista de tipo LayoutView, o sea compuesta. El esqueleto únicamente especifica las secciones que controla (con el objeto regions, que establece la correspondencia entre nombres simbólicos de secciones y nodos del HTML).

Fijate que en el archivo js/main.js, cuando se carga el documento (evento $(document).ready) se crea una instancia de la vista global y se muestra una vista con el formulario de búsqueda en la sección formBusqueda. Pero por ahora el formulario no hace nada. Vamos a solucionar esto enseguida.

5.2.6. VISTA DE BÚSQUEDA (1,25 PUNTOS)

Esta vista debe ser la encargada de mostrar el formulario, disparar la búsqueda y obtener los resultados, que le pasará a la vista global. Por ahora solo muestra el formulario.

Está representada en el código por:

- Una clase Marvel.Views.VistaBuscarComics, cuyo esqueleto básico ya tienes implementado en js/views/VistaBuscarComics.js
- Una ***template*** que tienes en index.html en forma de script con un id=VistaBuscarComicsTmpl, con un formulario básico (puedes modificarlo si lo deseas).

Para que esta vista haga su trabajo completo, tienes que:

- Definir el array events para que cuando se pulse sobre el botón con id=botonBuscar se llame a una función buscar de la vista, que debes definir.
- En esta función buscar debes llamar al método buscar de la colección (el que has probado antes en la consola).
- En el initialize debes crear la colección de comics asociada, por el momento vacía (se llenará de datos cuando se haga el fetch)

```
initialize: function() {
  this.collection = new Marvel.Collections.Comics();
  //Todavía faltan cosas
  ...
}
```

- Como la búsqueda es asíncrona, para saber cuándo se han recibido resultados usaremos el evento sync de la colección que indica que se ha recibido del servidor. En el initialize

debes hacer que la vista escuche el evento sync sobre la colección y que cuando se produzca llame a una función de la vista busquedaCompletada.

recuerda que cuando un evento llama a un ***callback***, this no apunta a la vista sino a window. Para solucionarlo puedes hacerlo de dos formas

```
//Forma 1: usando el bindAll de la librería underscore
_.bindAll(this, 'busquedaCompletada');
this.listenTo(this.collection, 'sync', this.busquedaCompletada);
//Forma 2 (USA SOLO UNA DE ELLAS!!): usando el "bind" estándar de Javascript
this.listenTo(this.collection, 'sync', this.busquedaCompletada.bind(this));
```

- Finalmente define la función busquedaCompletada para que lance un evento "a medida" (es decir, no estándar de Backbone) y que escuchará la vista global, luego veremos cómo.

```
busquedaCompletada: function() {
   //El nombre `completed:search` es totalmente inventado, podrías poner lo que quieras.
   //this.collection se le pasará como parámetro a quien esté escuchando este evento
   this.triggerMethod('completed:search', this.collection);
},
```

Sí, es un poco retorcido esto de capturar el evento sync para lanzar un evento completed:search. En Marionette una vista global puede escuchar eventos de las vistas hijas, pero no directamente de una colección gestionada por una vista hija. También podrías pasarle a la vista global una referencia a la colección para que pudiera escuchar directamente el evento sync. Pero es un poco embrollado que las vistas se pongan a escuchar eventos sobre objetos que en principio no son suyos.

5.2.7. De nuevo a la vista global (1,25)

Ahora para que la **vista global** escuche el evento completed:search y en respuesta muestre una vista con la lista de comics encontrados puedes añadirle lo siguiente:

```
childEvents: {
 //los eventos de las subvistas automáticamente reciben como 1er parámetro la subvista
 //y luego los que hayamos incluido cuando generamos el evento con el triggerMethod
 'completed:search' : function(child, col) {
    //Creamos una vista para mostrar una lista de comics, le pasamos la colección
    //y la mostramos en la sección "listado" de la vista global
    this.showChildView('listado', new Marvel.Views.VistaComics({
      collection: col
    }))
 }
}
```

5.2.8. Ver detalles de comic (1,25)

Nuestro objetivo ahora es que cuando pulsemos en "ver detalles" de un comic se sustituya la lista de comics por los datos detallados del comic elegido. Luego al "cerrar detalles" aparecerá de nuevo la lista.

- Lo primero, tendrás que modificar la ***template*** de la vista que solo muestra un comic VistaComic para añadirle un enlace o botón "ver detalles".
- Después usa la propiedad events de la vista para asociar el click sobre el enlace o botón con una función verDetalles

En la función verDetalles, lo primero asegúrate de que anulas el comportamiento por defecto del enlace o botón, ya que podría recargar la página y se comportaría de forma "extraña":

```
//Los manejadores de evento Javascript reciben automáticamente el evento producido
function verDetalles(evento) {
  //Anular el manejador por defecto del navegador, que podría recargar la página
  evento.preventDefault();
  ...
}
```

- Además de lo anterior, en la función verDetalles debes generar un evento "a medida" que llegará a la vista global y le indicará que hay que mostrar los detalles de un comic concreto. Pasa algo parecido a la búsqueda. En este caso una vista "madre" no puede escuchar directamente los eventos de interfaz de usuario de las hijas, así que las hijas tienen que capturarlos ellas mismas y lanzar un nuevo evento para la "madre". Finalmente verDetalles quedará:

```
//Los manejadores de evento Javascript reciben automáticamente el evento producido
function verDetalles(evento) {
  //Anular el manejador por defecto del navegador, que podría recargar la página
  evento.preventDefault();
  //me invento un evento (¡y rima!). Pasamos el modelo, para que la vista "madre" sepa
  //de quién hay que mostrar los detalles
  this.triggerMethod('show:details', this.model);
}}
```

Ahora tendremos que modificar el código de la vista global, que es la que tiene que recibir el evento show:details. En respuesta a este evento, sustituiremos la vista con la lista de comics por una vista para mostrar los detalles (clase VistaDetallesComic). Pero como cuando cerremos los detalles queremos volver a la lista, le diremos a Marionette que no destruya la vista de lista, y la guardaremos en el objeto vista global.

```
childEvents: {
  'completed:search' : function(child, col) {
    //...esto ya lo teníamos de antes
  },
  'show:details': function(child, model) {
     //guardamos la vista con el listado actual
     this.vistaLista = this.getRegion('listado').currentView;
     //Creamos una vista de tipo "detalle" asociada al modelo
     var nv = new Marvel.Views.VistaDetallesComic({model: model});
     //mostramos la nueva vista, diciéndole a Marionette que no libere la memoria
     //de la anterior, ya que luego la colocaremos otra vez en su sitio
     this.getRegion('listado').show(nv,{preventDestroy:true});
   }
```

```
}
```

Cuidado, si quieres probar esto, lo primero que debes hacer es rellenar la ***template*** asociada a la VistaDetallesComic, para que aparezca información en pantalla. Coloca los datos que quieras, y una imagen a mayor tamaño que la que aparece en el listado.

5.2.9. CERRAR LA VISTA DE DETALLES Y VOLVER AL LISTADO DE COMICS (1,25 PUNTOS)

En la ***template*** asociada a la clase VistaDetallesComic tiene que haber un botón o enlace "Cerrar". Ahora es similar al proceso seguido para mostrar los detalles. En la clase VistaDetallesComic

- Mediante el events debes asociar el click sobre "Cerrar" con una función de la vista cerrarDetalles
- En la función cerrarDetalles debes generar un evento "a medida" para que lo capture la vista "madre". Por similitud con el anterior, puedes llamarlo "hide:details" (o como quieras).

```
recuerda llamar a preventDefault() para evitar posibles recargas de la página.
```

Finalmente, modifica el código de la vista global para que reciba el evento hide:details (o como lo hayas llamado) y vuelva a poner "en su sitio" la lista de comics.

```
childEvents: {
  ...
  ...
  'hide:details': function() {
     this.getRegion('listado').show(this.vistaLista);
  }
}
```

5.3. REQUERIMIENTOS "ADICIONALES" (2 PUNTO EN TOTAL)

Con la guía anterior espero que hayas adquirido una idea básica de cómo coordinar varias vistas desde una vista "madre", que es la parte más complicada. Los requerimientos que faltan necesitan de la gestión de vistas y además de la interacción con el ***backend*** de Kinvey.

Pistas de implementación para los requerimientos que faltan:

Para el "login": *(0,7 puntos)*

Tendrás que definir una vista VistaFormLogin o similar que muestre un formulario de login. Tómate la operación de login simplemente como una "comprobación de que el login y el password son correctos". En realidad no usamos una sesión de usuario ya que recuerda que en cada petición a Kinvey mandamos de nuevo login y password, al estar usando HTTP Basic.

Para hacer login en Kinvey con el API REST hay que hacer una petición POST a https://baas.kinvey.com/user/MI_APP_ID/login, mandando un objeto JSON con las propiedades 'username' y 'password'. La petición debe estar autentificada usando HTTP BASIC con las credenciales de la aplicación, es decir, como login el APP_ID y como password el APP_SECRET, que aparecen en la parte superior del ***dashboard*** web de Kinvey.

puede haber cierta confusión entre las credenciales (login+password) del usuario y las credenciales de la aplicación. Para la operación de login en Kinvey debemos enviar un objeto JSON con las credenciales del usuario. Pero es que además TODAS las peticiones a Kinvey deben estar autentificadas. ¿Pero con qué credenciales, si precisamente estamos intentando hace login ahora mismo?. En Kinvey esto se resuelve usando las credenciales de la aplicación (APP ID+APP SECRET), que viene a ser una especie de "usuario global" de la app.

Crea un modelo Backbone llamado Usuario, con las propiedades username y password, y define en él un método login que llame internamente al save() de Backbone. Cuando el login se efectúe con éxito (evento sync sobre el modelo), la vista de la barra superior debería cambiar para mostrar tu login y un botón/enlace para ver "mis comics".

Aunque en Kinvey existe una operación de ***logout***, en nuestro caso no tiene sentido, porque no estamos usando sesiones. Simplemente cuando se haga ***logout*** tu aplicación debería dejar de mostrar los datos del usuario actual y volver a mostrar el formulario de login.

Para la gestión de "mis comics" *(1,3 puntos)*: La idea sería que usaras un modelo Favorito en la que almacenaras los datos que quieras guardar del comic (como mínimo su id, para poder recuperar el resto de datos con el API de Marvel, o bien copiar los campos y repetirlos en Kinvey para no tener que buscar de nuevo en Marvel). Tendrás que implementar:

- La parte de interfaz que te permite marcar un comic como favorito
- La sustitución de los resultados de búsqueda por la lista de tus comics cuando pulsas en "mis comics"

Como es lógico también deberían poderse eliminar los favoritos, pero ya son bastantes requerimientos, lo dejaremos fuera.

Puedes implementar estas funcionalidades de la forma que mejor te parezca, mientras funcionen correctamente.¡Suerte!.