# ¿Qué es Backbone.js?

## Que es Backbone.js?

Es una librería de JavaScript el cual nos ayudara a tener conexión a archivos json que contendrán el contenido a mostrar en nuestra página y lo estará actualizando constantemente mediante modelos y colecciones, a esto lo llamamos una librería de MV* (Modelo Vista pero no tenemos controlador) entonces de esta forma actualizaremos los json mediante el usuario interactúa con nuestra página y al tiempo creamos un estilo de API basada en json (porque nadie usa xml).

En la clase de Daniel usamos templates tanto del lado del server como del lado del cliente, te recomiendo leer esta lectura para seguir con el resumen.

Recuerda que no nos importa el server mientras nos arroje archivos JSON, en este caso usamos Node.js pero igual no es relevante para nosotros porque somos puro Frontend.

Lo que más nos importan son las rutas donde recogeremos los JSON.

***Pero, ¿WHAT? ¿Entonces el frontend hará todo?***
No, el backend se encargará de configurar el server para que te arroje los JSON, que reciba los eventos que manda el front y que guarde en base de datos, así que no debes infartarte ;)

Recuerda que los archivos con los que iniciamos la clase fueron los de PULS3 que hicimos con @freddier y en este caso entramos a terminar nuestra app.

Utilizaremos swig como template del lado del server nuestra app en donde en vez de repetir cada uno de los artículos lo sustituimos por un for  de cada post y generas un template dinámico y en base de variables, recuerda que hablamos de una página viva y debes de asumir esto para

empezar a usar los templates.

Del lado del cliente usaremos underscore.js y será la que muestre todo nuestro contenido del json y al manipular JSON como una API debemos de hacerlo en el cliente, PERO ESTO ES INSEGURO, lo sabemos y el backend es el que debe validar toda esta información y lidiar con la seguridad.

Ahora creamos nuestro archivo init.js en donde tenemos un namespace llamado puls3 que es un objeto con vistas colecciones modelos routers y utilidades de la siguiente manera.

```
window.Puls3 = {};
Puls3.Views = {};
Puls3.Collections = {};
Puls3.Models = {};
Puls3. Routers = {};
Puls3.util = {};
// Recuerda estos que acabamos de colocar son como namespaces
window.app ={};
window.routers = {};
window.plug = {};
window.views = {};
window.collections = {};
// y estas son el tipo de cosas que usaremos en nuestra app
```

Colocamos a init.js en nuestro index.html y vamos a el navegador y cargamos la página, el inspector de elemento será el mejor amigo del frontend profesional.

Con Grunt que es una librería que hace una librería automatizada la cual generará nuestros modelos de backbone con la siguiente línea:

```
> grunt bb_generate:model:article
```

En este caso el modelo se llama artículo.

Siempre que veas grunt piensa en algo automatizado, y bb en algo de backbone :)

Si vemos en el lugar donde ejecutamos el comando que debe ser el de js hay un folder llamado backbone con un folder llamado models con nuestro modelo generado automáticamente.

Lo incluimos a index.html y vamos a nuestra página y al inspector de elemento y colocamos:

```
> Puls3.Models.Article
```

Y aparece nuestro artículo.

Ahora este artículo será como un constructor entonces si colocamos la siguiente línea:

```
> m = new Puls3.Model.Article({title:”hola”})
```

Nos creará un artículo con el contenido colocado en los paréntesis y guardado en m

ahora si colocamos:

```
m.toJSON()
```

veremos el contenido de este.

Otra función buena del modelo es .get y .set en donde . get me obtendrá a base de un título y un .set que agregara un nuevo objeto al elemento, esto será esencial para el desarrollo de la app.

Algo importante de los modelos es que siempre los necesitamos con una colección porque una colección será ese banco de datos regidos por un modelos.

Entonces procedemos a generar una colección con grunt:

```
> grunt bb_generate:collection:articles
```

Si vamos a la carpeta js>backbone veremos que tenemos ahora una carpeta llamada collections y procedemos a cargar en el index nuestra colección generada.

y en el inspector de elementos crearemos una colección asi:

```
> c = new Puls3.Collections.ArticlesCollection()
```

Pero la creo vacía asi que le añadiremos información:

```
> m.add({title:'title'})
```

Ahora le diremos que esta se regirá por un modelo asi:

```
> m = c.first();
```

Y aca es muy importante porque aplicamos lo que explicaba anteriormente y es que ahora la colección se recibió información al estilo del modelo y se basara en este.

Recuerda que puedes jugar en el inspector con lo que acabamos de ver y con .toJSON ver cómo cambia nuestra colección o modelo.

O puedes colocar

```
> m.on('change', function (model) { console.log( model.toJSON() )})
```

Y con este cada vez que haga un cambio te imprimirá el modelo y veras el cambio realizado.

Recuerda que debes tener tu data de una manera accesible por consola lo cual hace que esto sea mucho más escalable al momento de mantener una app muy grande ya que cada pieza en tu app será muy chica pasando información entre eventos, esta es la clave y el paradigma del éxito al crear apps en JavaScript.

Ahora crearemos main.js en donde aplicaremos lo que acabamos de ver, entonces colocamos código que esté listo con document ready de jquery y hacemos request de ajax a el server para pedir la información.

```
var xhr = $.get('articles/all')
xhr.done (function (data) {
    console.log( data );
});
```

y veremos que nos carga nuestra información del server.

Ahora debemos guardarla en nuestra colección asi que primero la creamos fuera del xhr.done y la guardaremos en window.collection.articles.

```
window.collection.articles = new Puls3.Collections.ArticlesCollection ()
;
```

y dentro del xhr colocaremos un forEach al data y lo guardaremos uno a uno.

```
data.forEach (function ( item ){
    window.collections.articles.add(item);
});
```

Y ahora tenemos una colección con nuestra informacion del server agregada a nuestra app del lado del cliente.

Ahora tenemos que mostrar una vista por cada ítem, entonces generamos una vista llamada article.

```
> grunt bb_generate:view:article
```

y la debemos incluir en el index.html la cual se ubica en una carpeta views dentro de backbone.

Comentamos la linea 10 y 11 y en la línea 13 colocaremos dentro de los paréntesis de this.$el.html haya un this.model.get('title')

A las vistas lo primero que debemos pasarle es un modelo, entonces cada vista tiene un modelo y cada modelo tiene una colección.
En la línea 7 colocamos this.model = model y comentamos la linea 8.

Ahora le diremos que cuando se agregue algo a la colección se renderize el modelo que se agregó y esto lo hacemos con lo siguiente:

```
window.collections.articles.on ('add' ,function (model){
    var view = new Puls3.Views.ArticleView(model);
    view.render ();
    view.$el.insertAfter('#contenido aside');
});
```

y si vas a tu página automágicamente vemos texto :).

Ahora procedemos a crear un template mucho más bonito basándose en lo que ya hemos leído y haciendo de nuestra app la apariencia física a partir de JSON y el manejo de Backbone.js.

Recuerda que para actualizar un elemento debes guardar el modelo con

model.save() y actualizará el JSON así podremos manejar el tema de votos.

Espero practiques y lleves a Puls3 al siguiente paso de la clase 6 y crear cosas mucho más interesantes.

Repositorio oficial - Descarga .zip

Y como no presentarles la aplicación de PULS3 hecha en Angular.js por @melaspela, Denle una felicitación!