Las aplicaciones y páginas web están haciendo uso de código javascript cada vez en mayor cantidad para proporcionar aplicaciones con funcionalidades más complejas. En estos momentos las librerías para utilizarse en javascript son muchas y una aplicación puede hacer uso de varias de ellas.  
  
A pesar de las facilidades que proporcionan librerías como jQuery para manipular los elementos de una página web de forma dinámica, mucho código mal estructurado puede llegar a convertirse en un perfecto código espagueti, aunque jQuery facilita y simplifica muchas tareas en el manejo de los elementos de una página no es suficiente, a pesar de jQuery se puede llegar a tener un código con un mar de selectores, mantener sincronizados con la interfaz con el estado de la aplicación y la base de datos puede convertirse en una tarea complicada dar como resultado código con el que apetece poco trabajar.  
  
[Backbone.js](http://backbonejs.org/) trata de de resolver algunos de estos problemas proporcionando un marco sobre que el que organizar el código. Backbone.js en esencia en una librería modelo-vista-controlador (MVC) para javascript. Basándose en este patrón de diseño cada parte es independiente del resto. El modelo conserva el estado de la aplicación y produce eventos al ser modificado de forma que la vista se pueda actualizar de una forma independiente para ambos, la vista proporciona la interfaz del controlador a través del que se modifica el modelo de forma adecuada, todos los elementos están relacionados pero cada uno se encarga de una parte.



Backbone además proporciona una forma para trabajar con servicios REST tanto para recuperar los datos de los modelos como para actualizarlos, guardarlos y eliminarlos en el servidor. A continuación pondré un pequeño ejemplo usando backbone.js que calcula el área de un rectángulo dado su alto y ancho y que lo muestra en pantalla. Como backbone.js sigue el patrón de diseño MVC por un lado tendremos el modelo, en nuestro caso el rectángulo con las propiedades alto y ancho que nos servirán para calcular el área, por el otro la vista que se actualizará según los datos del modelo introducidos en dos campos de texto, finalmente el controlador que captura los eventos como los cambios en los campos de texto, modifica el modelo y el modelo notifica a la vista para que se actualice.  
  
Vemos un ejemplo simple y que muestra parte del potencial de backbone.js. El modelo se define con el siguiente código, además de las propiedades se puede definir funciones que nos puede ser de utilidad, como calcular el área o obtener un objeto para ser utilizado con Mustache:

|  |  |
| --- | --- |
| 1234567891011121314 | var Rectangulo = Backbone.Model.extend({  defaults: {  alto: 4,  ancho: 3,  },  area: function() {  return this.get('alto') \* this.get('ancho');  },  toTemplateJSON: function() {  var json = this.toJSON();  json.area = this.area();  return json;  }  }); |

[**view raw**](https://gist.github.com/picodotdev/5421633/raw/main-1.js) [**main-1.js**](https://gist.github.com/picodotdev/5421633#file-main-1-js) hosted with ❤ by [**GitHub**](https://github.com/)

<a href="https://gist.github.com/picodotdev/5421633/raw/main-1.js">main-1.js</a><br/>   
La otra parte es la vista y el controlador, ambas cosas se definen con Backbone.View.extend:

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334 | var Vista = Backbone.View.extend({  el: '#area',  events: {  "change input[name='alto']": 'onChangeAlto',  "change input[name='ancho']": 'onChangeAncho'  },    initialize: function() {  \_.bindAll(this, 'render', 'onChangeAlto', 'onChangeAncho');    this.model = new Rectangulo();  this.model.on('change', this.render);    this.render();  },    onChangeAlto: function() {  var v = parseInt($("input[name='alto']", this.el).val());  this.model.set('alto', v);  },    onChangeAncho: function() {  var v = parseInt($("input[name='ancho']", this.el).val());  this.model.set('ancho', v);  },    render: function() {  $("input[name='alto']", this.el).val(this.model.get('alto'));  $("input[name='ancho']", this.el).val(this.model.get('ancho'));    var texto = Mustache.render('El área de un rectángulo de alto {{alto}} y ancho {{ancho}} es <b>{{area}}</b>', this.model.toTemplateJSON());  $('#resultado', this.el).html(texto);  },  }); |

[**view raw**](https://gist.github.com/picodotdev/5421633/raw/main-2.js) [**main-2.js**](https://gist.github.com/picodotdev/5421633#file-main-2-js) hosted with ❤ by [**GitHub**](https://github.com/)

<a href="https://gist.github.com/picodotdev/5421633/raw/main-2.js">main-2.js</a><br/>La inicialización del ejemplo:

|  |  |
| --- | --- |
| 1 | var v = new Vista(); |

[**view raw**](https://gist.github.com/picodotdev/5421633/raw/main-3.js) [**main-3.js**](https://gist.github.com/picodotdev/5421633#file-main-3-js) hosted with ❤ by [**GitHub**](https://github.com/)

<a href="https://gist.github.com/picodotdev/5421633/raw/main-3.js">main-3.js</a><br/>   
Las partes de esta vista-controlador son:

* el: el elemento html sobre el que actuará la vista.
* events: los eventos que manejará el controlador, es un mapa donde la clave es un selector muy parecido a jquery y el nombre de la función a llamar cuando se produzca.
* initialize: con la función de underscore \_.bindAll haremos que la referencia a this sea la vista en las funciones indicadas (render, onChangeAlto, onChangeAncho). A continuación se crea el modelo y cuando se produzca algún cambio en él (evento change) se actualice la vista. Finalmente, cuando se inicia la aplicación se dibuja por primera vez la vista.
* onChangeAlto y onChangeAncho: obtienen el alto o ancho de los campos de texto y modifican el modelo.
* render: se encarga de actualizar la vista cuando haya algún cambio en el modelo tal y como hemos echo en la función initialize con la función on sobre el modelo. Actualiza los inputs de alto y ancho con los valores del modelo y mediante una plantilla de Mustache forma el mensaje que se visualiza en el  elemento con id resultado.

Finalmente, queda el pequeño código html de la aplicación que contiene los campos de texto para introducir el alto y ancho y el elemento div donde se mostrará el área calculado del rectángulo:

|  |  |
| --- | --- |
| 123456789 | <div id="area">  <label for="alto">Alto</label>  <input id="alto" name="alto" value="">    <label for="ancho">Ancho</label>  <input id="ancho" name="ancho" value="">    <div id="resultado" style="margin-top: 2em"></div>  </div> |

[**view raw**](https://gist.github.com/picodotdev/5421633/raw/HelloWorldBackbone.html) [**HelloWorldBackbone.html**](https://gist.github.com/picodotdev/5421633#file-helloworldbackbone-html) hosted with ❤ by [**GitHub**](https://github.com/)

<a href="https://gist.github.com/picodotdev/5421633/raw/HelloWorldBackbone.html">HelloWorldBackbone.html</a><br/>

[](http://3.bp.blogspot.com/-okBxtRGOpAs/UXGHhhKIm1I/AAAAAAAACeo/zRjC-0Oo4aw/s1600/helloworldbackbone.png)

Backbone.js no es la única librería que implementa el patrón MVC en javascript otras alternativas son [knockout](http://knockoutjs.com/) y [AngularJS](http://angularjs.org/) aunque esta últimas tienen la notable diferencia de que se basan en añadir atributos especiales a las etiquetas html para definir el modelo y asociar los eventos. Backbone.js no necesita añadir esos atributos especiales a la etiquetas html o «instrumentalizarlo», esto se hace a través de javascript lo que da una mayor separación entre el código html y el código javascript. Por ahí se dice que usando backbone.js para realizar lo mismo se necesitan más lineas de javascript pero hay que tener en cuenta que a cambio el html necesario es mucho más simple que en knockout o AngularJS lo que facilita que el trabajo con motores de plantillas como [Mustache](http://mustache.github.io/) o [Handlebars](http://handlebarsjs.com/) también sea más simple.  
  
Este [pequeño manual de backbone](http://desarrolloweb.com/manuales/manual-backbonejs.html) explica muy bien que es. Y el siguiente vídeo es muy interesante.



Como en el resto de entradas sobre esta serie de javascript el código fuente del ejemplo completo lo puedes encontrar en mi [repositorio de GitHub](https://github.com/picodotdev/elblogdepicodev/tree/master/HelloWorldBackbone). Este ejemplo se basa en [RequireJS](http://elblogdepicodev.blogspot.com.es/2013/03/introduccion-y-ejemplo-de-requirejs.html) y [Mustache](http://elblogdepicodev.blogspot.com.es/2013/03/introduccion-y-ejemplo-de-mustache.html) visto en anteriores entradas que puedes consultar en el apartado de referencia al final de la entrada.  
  
En la siguiente entrada mostraré [un ejemplo más completo y usando muchas cosas al mismo tiempo](http://elblogdepicodev.blogspot.com.es/2013/04/ejemplo-lista-de-tareas-con-backbone.html) de las que he estado hablando individualmente en las últimas entradas (ver referencia), será un ejemplo de una aplicación de lista de tareas por hacer que permite crearlas y completarlas, usará RequireJS, Mustache y Backbone (usándolo de forma más avanzada y completa de lo visto en esta entrada) en la parte cliente y [Tapestry](http://tapestry.apache.org/) y [RESTEasy](http://www.jboss.org/resteasy/) en la parte servidor.