HANDLEBARS

Tiempo estimado: 25min

En la lección anterior, introdujimos los motores de plantillas, pero sin describir ninguno en particular, sólo el soporte por parte de Express. En esta lección, vamos a describir Handlebars, ampliamente utilizado tanto en Express como en otras aplicaciones con más de cuatro millones de descargas mensuales.

Comenzamos introduciendo, instalando y configurando Handlebars como motor de plantillas de la aplicación. A continuación, seguimos con las plantillas, cómo crearlas y qué elementos usar para añadir contenido dinámico. Finalmente, presentamos cómo reutilizar plantillas Handlebars mediante el concepto de parcial y cómo añadir nueva funcionalidad a Handlebars mediante las funciones asistentes.

Al finalizar la lección, el estudiante sabrá:

- Cómo instalar y configurar el motor de plantillas Handlebars para su uso en Express.
- Cómo crear plantillas Handlebars.
- Cómo definir funciones asistentes para añadir funcionalidad a Handlebars.
- Cómo configurar nodemon para que monitorice los cambios de las plantillas Handlebars.

INTRODUCCIÓN

Handlebars es un motor de plantillas muy sencillo. Sus plantillas son archivos de texto como, por ejemplo, páginas HTML e incluso texto plano, con secciones que añaden información dinámicamente cada vez que el motor procesa las plantillas. Se basa en el motor Mustache y en él se basa Spacebars de Meteor.

Su página web oficial handlebarsjs.com.

Se ha elegido Handlerbars como motor de plantillas por las siguientes razones:

- Es ampliamente aceptado y utilizado.
- Es rápido.
- Está ampliamente probado.
- Está implementado tanto para cliente como para servidor.
- En él se basa Spacebars, el motor de plantillas nativo de Meteor, una plataforma basada en Node para el desarrollo de aplicaciones webs clientes y servidoras. También es el motor de plantillas nativo del *framework* Ember.js y de la herramienta de automatización Justo.js.

  Así, si el estudiante decide ampliar conocimientos y estudiar Meteor, Ember.js o Justo.js ya tendrá un paso dado.
- Se basa en Mustache, otro motor de plantillas ampliamente reconocido y consolidado.

  Si el estudiante ya conoce Mustache, le resultará bastante fácil aprender Handlebars.
- Es extensible. Permite añadir nueva funcionalidad muy fácilmente.

INSTALACIÓN DEL MOTOR HANDLEBARS

Para instalar el motor de plantillas Handlebars en una aplicación Express, hay que usar el paquete hbs. Hay más paquetes, pero éste es el recomendado por el equipo de Express. Debemos incorporarlo a las dependencias de la aplicación, mediante la propiedad dependencies del archivo package.json:

```
"dependencies": {
  "express": "*",
  "hbs": "*",
```



```
}
```

Configuración del motor de plantillas Handlebars

No hay que olvidar configurar Handlebars como el motor predeterminado de la aplicación. Primero, tenemos que tener claro la extensión a usar para las plantillas que, por convenio, es .hbs. Y a continuación, hay que indicarle a Express que el motor de plantillas para esta extensión es Handlebars.

```
app.set("view engine", "hbs");
app.engine("hbs", require("hbs").__express);
```

No olvidemos configurar la carpeta de plantillas mediante la opción views. Recordemos, el lugar al que debe ir render() a buscarlas:

```
app.set("views", path.join(__dirname, "views"));
```

Plantillas Handlebars

Una plantilla Handlebars (*Handlebars template*) es un texto que contiene código estático y dinámico a través del cual añadir datos a la reproducción. Permite añadir información dinámicamente como, por ejemplo, los datos de una orden de pedido o los datos de un empleado. El resultado del procesamiento de la plantilla será el duplicado que se remitirá como respuesta al cliente. Por convenio, las plantillas Handlebars tienen la extensión .hbs.

Para ir abriendo boca, veamos un ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>Welcome to {{title}}</p>
  </body>
</html>
```

En este ejemplo, la plantilla contiene código HTML y expresiones Handlebars a través de las cuales añadir los datos dinámicamente a la página web. Lo delimitado por {{ y }}.

Comentarios

Un comentario (*comment*) es un texto que no tiene ningún significado gramatical para el compilador de plantillas. Son muy útiles porque ayudan a describir qué hace un determinado fragmento de código. Los comentarios no formarán parte del duplicado final. Se indican como sigue:

```
{{!-- comentario --}}
{{! comentario }}
```

Expresiones en línea

Una expresión (*expression*) es un comando u operación interna del motor de plantillas. En el caso de Handlebars, se delimitan mediante llaves, principalmente dobles, {{ y }}. Un tipo especial de expresión es la expresión en línea (in-line *expression*), aquella cuya evaluación devuelve un valor, el cual se inserta en el punto en el que se encuentra dentro de la plantilla, distinguiéndose entre las que escapan el valor devuelto por la expresión y las que no lo hacen.

Expresiones de escape

Una expresión de escape (*escape expression*) es aquella que escapa el valor devuelto a HTML. Se especifican entre llaves dobles:

```
{{expresión}}
```

No olvidemos que cuando el motor de plantillas evalúa la expresión, cualquier identificador que aparezca dentro de ella se buscará en el contexto de ejecución.

El siguiente ejemplo muestra cómo añadir el valor de la propiedad title del contexto actual en un elemento HTML <h1>:


```
<h1>{{title}}</h1>
```
Se puede acceder a cualquier propiedad del contexto, de manera similar a como lo hacemos en JavaScript. Un nuevo ejemplo:

```
<h1>{{scope.name}}</h1>
```

Expresiones seguras

Una expresión segura (*safe expression*) es aquella que añade el valor devuelto por la expresión tal cual al duplicado, sin escapar nada. Se delimitan entre llaves triples:

```
{{{expresión}}}
```
¿Qué significa que una expresión segura no escapa y una de escape sí lo hace? Todo está relacionado con HTML. Mediante una expresión de escape, la delimitada por {{ y }}, el motor analizará el resultado de la expresión y, si es necesario, reemplazará los caracteres especiales como <, >, etc., por entidades HTML como < y >. En cambio, con una expresión segura no hará nada de eso y añadirá el valor de la expresión tal cual al resultado.

Expresiones de bloque

Una expresión de bloque (*block expression*) es otro tipo especial de expresión con la que se inserta un determinado fragmento de código plantilla cero, una o más veces en el punto en el que se encuentra. Este tipo de expresiones se delimita mediante un par de llaves con una almohadilla (#):

```
{{#nombre}}
  código de bloque
{{/nombre}}
```
Observe que el nombre del inicio de la expresión lleva una almohadilla (#) y el cierre una barra (/).

Se distingue entre iteradores y condicionales.

Condicionales

Un condicional (*conditional*) es un bloque que presenta un fragmento plantilla u otro atendiendo a una condición. Tiene como objeto implementar la sentencia if de JavaScript. Su sintaxis es:

```
{{#if condición}}
  cuerpo si se cumple la condición
{{else}}
  cuerpo en otro caso
{{/if}}
```
La sección {{else}} es opcional.

He aquí un ejemplo ilustrativo:

```
{{#if scope.article.inStock}}
  <img src="inStock.gif" alt="In Stock">
{{/if}}
```
También se puede utilizar la expresión de bloque {{#unless}} cuyo comportamiento es similar a {{#if}} pero desde el punto de vista contrario. O sea, en un bloque {{#if}} , el {{#if}} tiene en cuenta su sección si su condición se cumple, en otro caso, reproduce el {{else}}. En el caso de {{#unless}}, su sección se seleccionará cuando la condición *no* se cumpla y {{else}} si se cumple:

```
{{#unless condición}}
  cuerpo si NO se cumple la condición
{{else}}
  cuerpo en otro caso
{{/unless}}
```

Iteradores

Un iterador (*iterator*) es un bloque que repite cero, una o más veces un determinado fragmento plantilla para cada elemento devuelto por una expresión. Su sintaxis es la siguiente:

```
{{#each variableIteración [as |valor clave|]}}
  cuerpo de iteración
{{else}}
  cuerpo si no hay nada para iterar
{{/each}}
```


La expresión devuelve los elementos sobre los que iterar, de manera muy similar a la sentencia for de JavaScript. Para cada uno de los elementos de iteración, el cuerpo del bloque se duplica, pudiendo utilizar en cada duplicado el valor de iteración almacenado en la variable de iteración. El bloque {{else}} es opcional y, si se especifica, contiene lo que se duplicará si la expresión de iteración no devuelve nada con lo que iterar..

La cláusula opcional as se utiliza para añadir variables locales al contexto donde almacenar el valor y la clave de iteración actuales. Si el elemento de iteración es un *array*, la clave contendrá el índice en curso; y el valor, el elemento. En cambio, si es un objeto: la clave la propiedad en iteración; y valor, su valor.

Por ejemplo, supongamos que disponemos de un *array* de objetos o simplemente un objeto. Veamos un ejemplo ilustrativo que nos ayudará a comprenderlo mejor:

```
{{!-- scope.bands = [{…}, {…}, {…}] -}}
{{#each scope.bands}}
  <article>
    <h2>{{name}}</h2>
    {{{bio}}}
  </article>
{{/each}}
```

Los iteradores se ejecutan con su propio contexto, el cual está formado por el elemento de la iteración actual. En el ejemplo anterior, se asume que scope.bands devolverá un *array* de objetos con la información de las bandas. Para cada una de ellas, se creará un contexto y se generará el contenido de la plantilla para el elemento en cuestión. Como el contexto es el elemento de iteración, las variables locales utilizadas en las expresiones de plantilla harán referencia a las propiedades del elemento de iteración. Así pues, {{name}} hace referencia a la propiedad name de la banda en procesamiento; y bio a la propiedad homónima.

Para especificar explícitamente el contexto actual, se puede utilizar la referencia this. El ejemplo anterior podría escribirse como sigue:

```
{{!-- scope.bands = [{…}, {…}, {…}] --}}
{{#each scope.bands}}
  <article>
    <h2>{{this.name}}</h2>
    {{{this.bio}}}
  </article>
{{/each}}
```

O bien:

```
{{!-- scope.bands = [{…}, {…}, {…}] --}}
{{#each scope.bands as |band ix|}}
  <article>
    <h2>{{band.name}}</h2>
    {{{band.bio}}}
  </article>
{{/each}}
```

Para gustos, colores.

Ejecución con contexto específico

Se puede utilizar el bloque with para indicar que un determinado bloque plantilla se ejecute con un determinado contexto. Su sintaxis es como sigue:

```
{{#with objetoContexto [as |variable|]}}
  bloque a ejecutar con el objeto indicado como contexto
{{else}}
  bloque a ejecutar si el objeto indicado está vacío
{{/with}}
```

Ejemplo:

```
{{#with post}}
  <h1>{{title}}</h1>  {{!-- similar a post.title --}}
  {{{message}}}       {{!-- similar a post.message --}}
{{/with}}
```


Variables de datos

El motor de Handlebars dispone de un conjunto de variables predefinidas, de sólo lectura, a través de las cuales pasar datos a la reproducción relacionados con el contexto. Estas variables se pueden acceder mediante una arroba (@) seguida del nombre de la variable. He aquí las variables de datos más utilizadas:

Variable	Tipo de datos	Descripción
@root	Object	Contexto raíz.
@first	Boolean	En un iterador, indica si estamos ante la primera iteración del bloque: true, sí; false, no.
@last	Boolean	En un iterador, indica si estamos ante la última iteración del bloque: true, sí; false, no.
@index	Number	En un iterador, indica el índice de iteración, comenzando en cero.
@key	Object	En un iterador en el que se itera sobre un objeto, el nombre de la propiedad actual en iteración.

A continuación, un ejemplo:

```
{{#each posts}}
  <h2>Post nº {{@index}}: {{title}}</h2>
  {{message}}
{{/each}}
```

Reutilización de plantillas

La reutilización es un concepto necesario en el desarrollo de software y no podía faltar en Handlebars. Se puede utilizar parciales como medio de reutilización de componentes o subplantillas, según se quiera ver. Y también se puede utilizar las plantillas de diseño como una plantilla que contiene el aspecto base heredado por las plantillas.

Parciales

Un parcial (*partial*) es una subplantilla especializada en un determinado aspecto. Algo así como un elemento reutilizable. Las plantillas pueden incorporar parciales, facilitando así la reutilización.

Si se hace uso de parciales, por convenio, hay que depositarlos en el directorio de parciales (*partial directory*), donde cada archivo de este directorio representa un parcial. De manera predeterminada, el directorio de parciales se encuentra en views/partials. Podemos registrar los parciales mediante la función registerPartials() de hbs:

```
function registerPartials(dir)
```

Parámetro	Tipo de datos	Descripción
dir	string	Directorio de parciales.

Ejemplo:

```
hbs.registerPartials(path.join(__dirname, "views/partials/"));
```

Por ejemplo, en un blog, se podría definir un parcial como plantilla de un determinado *post*, de un determinado comentario o de la lista de comentarios de un *post*. Pudiendo reutilizarlo donde sea necesario dentro de la aplicación.

Inclusión de parciales

La inclusión de parciales (*partial include*) es la operación mediante la cual se incorpora un parcial a una plantilla. Se utiliza una expresión de inclusión (*include expression*):

```
{{> parcial}}
```

El nombre del parcial es el nombre del archivo en el que se define. Así, por ejemplo, si se desea incorporar el parcial views/partials/customer.hbs, utilizaremos una expresión de inclusión como la siguiente:

```
{{> customer}}
```

Cuando se procesa el parcial, Handlebars utiliza como contexto el actual.

Plantillas de diseño

Una plantilla de diseño (*layout template*) es una plantilla base que define el esqueleto de los duplicados

generados mediante plantillas Handlebars. Esta plantilla debe tener un marcador (*place holder*), que indica dónde ubicar el contenido de la plantilla duplicada. Se especifica mediante la expresión segura {{{body}}}.

He aquí un ejemplo de plantilla de diseño:

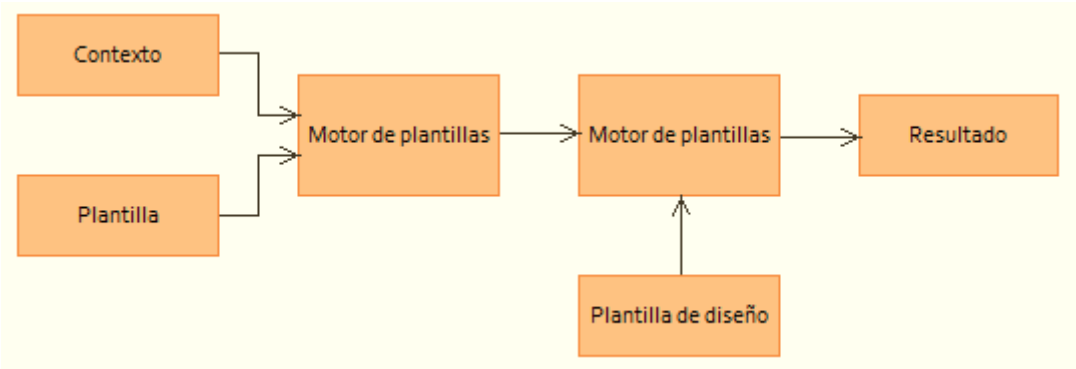
```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
  </head>

  <body>
    {{{body}}}
  </body>
</html>
```

Y ahora una plantilla estándar, cuyo resultado se ubicará en el marcador {{{body}}} de la plantilla de disposición:

```
<p>
  Banda: {{scope.name}}<br>
  Origen:  {{scope.origin}}<br>
  Año: {{scope.year}}
</p>
```

Las plantillas de diseño se utilizan principalmente para definir el aspecto general de las páginas webs. Se define todo aquello que es común a todas las vistas en una plantilla base, la de diseño, indicando en qué punto de esta plantilla hay que ubicar el contenido específico de cada plantilla. Cuando el motor de plantillas procesa una plantilla, lo que hace es reproducir, por un lado, la plantilla solicitada y, por otro lado, la plantilla de diseño. El resultado final será el resultado de la plantilla de diseño, con el contenido de la otra plantilla en el lugar en el que se encuentra el marcador {{{body}}}.



Plantilla de diseño predeterminada

Toda aplicación puede tener una plantilla de diseño predeterminada (*default layout template*), aquella que se utiliza si no se especifica otra explícitamente. Aquella que tiene el nombre layout. Siguiendo nuestros convenios, sería la definida en views/layout.hbs.

Utilización de plantilla de diseño específica

En algunas ocasiones, es necesario utilizar una plantilla de diseño distinta a la predeterminada. En la invocación del método render() de la respuesta, indicar el nombre de la plantilla en la propiedad layout. A continuación, se muestra un ejemplo ilustrativo:

```
app.get("/page", function(req, res) {
  res.render("template", {layout: "layout2", title: "…", scope: {…}});
});
```

La variable local layout del contexto tiene un significado especial para el motor de plantillas. No se recomienda definirla sino es para indicar explícitamente la plantilla de diseño a utilizar, en lugar de la predeterminada.

Funciones asistentes

Una función asistente (*helper function*) es una función que puede invocarse como una expresión en línea o bloque. Son un medio mediante el cual extender la lógica o funcionalidad de Handlebars. Son como funciones globales a todas las plantillas, pues se definen a nivel de aplicación.

Registro de funciones asistentes

Para poder utilizar una función asistente en una plantilla, es necesario registrarla mediante la función registerHelper() del módulo hbs:

```
function registerHelper(name, fn)
```

Parámetro	Tipo de datos	Descripción
name	string	Nombre de la función.
fn	function	Función *helper*: fn(...params).

A continuación, vamos a centrarnos más en cómo definir cada uno de los tipos de función asistente.

Funciones asistentes de bloque

Una función asistente de bloque (*block helper function*) es aquella cuya invocación se delimita por {{#nombre}} y {{/nombre}}. Al igual que {{#if}} o {{#each}}. Estas funciones se definen, tal como acabamos de ver, mediante la función registerHelper() del módulo hbs. Y tienen la siguiente signatura:

```
function helper(param1, param2, param3… opts)
```

Parámetro	Tipo de datos	Descripción
paramX	object	Parámetro(s) específico(s) de la función.
opts	object	Opciones de bloque: • fn (function). Función equivalente. • Inverse (function). Función inversa.

Los primeros parámetros de las funciones asistentes son parámetros a través de los cuales se puede pasar argumentos específicos a la función. El último siempre es un objeto de opciones específico de Handlebars. Este último parámetro se usa para indicarle al motor de plantillas qué debe reproducir. Vamos a precisar más.

Supongamos que tenemos una función asistente que reproduce un bloque si dos valores son iguales. Algo así como:

```
{{#eq scope.name 'Augustines'}}
  código a reproducir si se cumple que scope.name es igual a Augustines
{{/eq}
```

Como se puede observar, la función tiene dos parámetros específicos, los que contienen los valores a comparar mediante igualdad. Si se cumple la condición, deberá reproducirse el bloque asociado a {{#eq}}, en cualquier otro caso, no. ¿Cómo se implementa esta función? Muy fácil:

```
hbs.registerHelper("eq", function(val1, val2, opts) {
  return val1 == val2 ? opts.fn(this) : opts.inverse(this);
});
```

Desglosemos... Por un lado, tenemos tres parámetros: val1, val2 y opts. Los dos primeros son específicos de la función. Si la función tuviera ocho, pues definiríamos ocho. La cuestión es que como último parámetro, tendremos siempre opts, el cual es un objeto con dos funciones especiales: fn() e inverse().

La función equivalente (*equivalent function*), esto es, opts.fn(), devuelve el contenido del bloque definido entre {{#eq}} y {{/eq}}. Mientras que la función inversa (*inverse function*), o sea, opts.inverse(), para entendernos, como un indicador de *nada*. Cuando invoquemos estas funciones no debemos olvidar pasarles el argumento this. Así pues, cuando deseemos reproducir el contenido del bloque, la función asistente debe devolver el valor devuelto por la función equivalente, opts.fn(this). Mientras que si no deseamos que se reproduzca, el valor devuelto por la función inversa, opts.inverse(this).

Parece complicado, pero no lo es. Los puntos claves a recordar son:

- La función asistente de bloque debe devolver el contenido a reproducir.
- La función puede definir cualquier número de parámetros, siendo el último uno interno y específico de Handlebars.
- Para devolver el cuerpo del bloque, se usa opts.fn(this).

- Para indicar que no se reproduzca el contenido del cuerpo del bloque, se usa opts.inverse(this).

Funciones asistentes en línea

También es posible definir funciones asistentes en línea (in-line *helper functions*), aquellas que se invocan como una expresión en línea mediante {{ y }} como, por ejemplo:

```
{{suma 1 2 3 4}}
```

El resultado de la función es lo que se reproduce o se utiliza si se usa en una condición como, por ejemplo, en la condición del bloque {{#if}}. En este caso, las funciones deben presentar la siguiente signatura:

```
function(param1, param2, …, paramN)
```

A diferencia de las funciones asistentes de bloque, el último parámetro *no* es opts. El resultado devuelto por la función será reproducido o utilizado por el motor para realizar su trabajo. He aquí un ejemplo de una función asistente en línea que devuelve si dos valores son iguales:

```
hbs.registerHelper("eq", function(val1, val2) {
  return val1 == val2;
});
```

He aquí un ejemplo de su uso:

```
{{eq scope.name 'Vampire Weekend'}}
```

Y aquí uno que utiliza la función en un bloque {{#if}}:

```
{{#if (eq scope.name 'Vampire Weekend')}}
  …
{{/if}}
```

Para invocar una función asistente en línea, hay que indicar su nombre seguido de su lista de argumentos separados por espacios en blanco:

```
función
función argumento
función argumento argumento…
```

Subexpresiones

Una subexpresión (*subexpression*) es una expresión definida dentro de otra. En vez de delimitarse mediante llaves, se delimita mediante paréntesis:

```
( expresión )
```

Se utilizan principalmente para invocar funciones asistentes en línea, tal como vimos en el ejemplo anterior del bloque {{#if}}.

He aquí un ejemplo ilustrativo:

```
{{{fn (subfn arg) arg}}}
```

nodemon

De manera predeterminada, nodemon monitoriza el directorio actual y dentro de él los archivos con las extensiones .js, .json, .coffee y .litcoffee. Esto significa que *no* monitoriza las plantillas de Handlebars. Si necesitamos que se reinicie la aplicación si alguna plantilla es modificada, podemos indicarle que lo haga mediante la opción -e o --ext:

```
-e lista-separada-por-comas-sin-espacios
--ext list-separada-por-comas-sin-espacios
```

Veamos un ejemplo mediante el cual indica que se monitorice los archivos con las extensiones .hbs, .js y .json:

```
nodemon -e hbs,js,json ./app.js
```

