INTRODUCCIÓN A LAS PLANTILLAS

Tiempo estimado: 15m

En esta lección, se presenta uno de los tres componentes claves de Express, el motor de plantillas, el cual facilita la generación de contenido dinámico. Indispensable para muchas aplicaciones de hoy en día. En este lección, damos una pincelada inicial a las plantillas y, en la siguiente, una descripción detallada del motor de plantillas Handlebars.

Para comenzar, se introduce el concepto de capa de motor de plantillas. Se continúa mostrando cómo se procesan las plantillas, haciendo hincapié en el procesamiento, los contextos de ejecución y el ámbito. Finalmente, enseñamos cómo configurar los motores de plantillas usados en la aplicación.

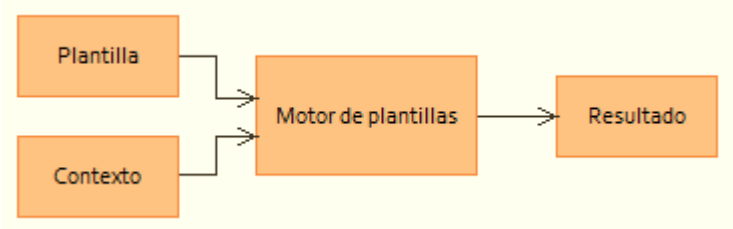
Al finalizar la lección, el estudiante sabrá:

- Qué es un motor o sistema de plantillas.
- Cómo configurar la carpeta donde se alojan las plantillas de la aplicación.
- Cómo solicitar el procesamiento de una plantilla.
- Qué es el contexto y el ámbito.
- Cómo configurar los motores de plantillas utilizados en la aplicación.

INTRODUCCIÓN

Un motor de plantillas (*template engine*) o sistema de plantillas (*template system*) es un componente que se encarga de procesar plantillas (*template*), también conocidas en Express como vistas (*views*), no son más que archivos de texto a partir de los cuales generar otros archivos dinámicamente como, por ejemplo, páginas HTML, archivos JSON y documentos de texto. Una plantilla tiene un fragmento de texto estático que no sufre variaciones, tal cual aparece en la plantilla, es tal como se remite al cliente. Pero también contiene texto que debe analizarse con cada petición, aquel que permite insertar datos dinámicos procedentes, por ejemplo, de una base de datos, servicio REST o de un parámetro.

Se conoce formalmente como reproducción (*render*) a la operación mediante la cual se invoca el motor de plantillas para reproducir una plantilla y, así, obtener un duplicado (*duplicate*). Cada ver que se realiza una reproducción, se crea un contexto de ejecución (*execution context*), o simplemente contexto (*context*), esto es, un conjunto de parámetros (*parameters*) o variables locales (*local variables*) que contiene datos específicos de esa reproducción.



Dada una plantilla y un contexto, el motor realiza el procesamiento y acaba generando el resultado a remitir al cliente.

Existe multitud de motores de plantillas como, por ejemplo, EJS, Handlebars, Mustache y Pug. A la hora de elegir un motor de plantillas, es importante tener claro principalmente los siguientes aspectos:

- Que permita la generación de recursos webs tanto HTML como de texto plano.
- Que sea rápido y óptimo. Cuanto más lento sea, más tiempo tardará en remitirse la vista al usuario. Y mayor será la latencia.

- Que esté probado y consolidado dentro de la comunidad.

A continuación, se muestra un ejemplo de plantilla EJS:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

Ahora, el mismo ejemplo pero mediante una plantilla Handlebars:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>Welcome to {{title}}</p>
  </body>
</html>
```

Observe que cada plantilla añade código dinámicamente de una manera distinta. Pero en ambos ejemplos, el objetivo final es obtener dinámicamente una página web como la siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express</title>
    <link rel='stylesheet' href='stylesheets/style.css' />
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>Welcome to Express</p>
  </body>
</html>
```

Carpeta de plantillas

La carpeta de vistas (*view folder*) es la carpeta en la que se encuentran las plantillas. Por convenio y buenas prácticas, se ubican en la carpeta /views o /templates del directorio de la aplicación. Tal como veremos más adelante en esta lección, cuando deseemos procesar una plantilla, indicaremos su nombre o ruta relativa a esta carpeta.

Configuración de la carpeta de vistas

Para configurar cuál es el directorio de la aplicación que contiene las plantillas, se utiliza la opción views:

```
app.set("views", folder)
```

Parámetro	Tipo de datos	Descripción
folder	string	Ruta a la carpeta de vistas.

Ejemplo:

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

Caché de vistas

La caché de vistas (*view cache*) es una caché interna en la que Express guarda las plantillas más utilizadas por la aplicación para mejorar así su rendimiento al evitar tener que releerlas del disco.

De manera predeterminada, la caché de vistas viene activada y se configura mediante la opción view cache:


```
//activación
app.enable("view cache");

//desactivación
app.disable("view cache");
```

Durante el desarrollo, se recomienda mantenerla desactivada. En producción, activada para mejorar el rendimiento. He aquí un ejemplo de configuración:

```
if (app.get("env") == "production") {
  app.enable("view cache");
} else {
  app.disable("view cache");
}
```

Procesamiento de plantillas

Las plantillas no se procesan implícitamente. Hay que solicitarlo explícitamente mediante el método render() de la respuesta. Además, es muy importante conocer cómo Express procesa las plantillas. Primero, se busca la plantilla en la carpeta de plantillas. A continuación, la aplicación determina cuál de los motores está asociado a la extensión de la plantilla. Después, genera el contexto bajo el cual se debe procesar la plantilla; no es más que el objeto que contiene los datos a utilizar durante el procesamiento. Finalmente, con la plantilla en una mano y el contexto en la otra, invoca el motor de plantillas y espera que le devuelva el resultado a remitir al cliente.

Método response.render()

Recordemos que cuando se recibe una petición HTTP, Express genera básicamente dos objetos: uno que representa y contiene la información de la petición HTTP remitida por el cliente. Y otro que representa y contiene la respuesta HTTP que se enviará como contestación. Pues el objeto respuesta contiene un método mediante el cual le informamos a Express que le solicite al motor de plantillas que procese una determinada plantilla, con un determinado contexto, siendo su resultado lo que debe añadir a la respuesta a remitir.

render() tiene un comportamiento muy parecido a sendFile(). sendFile() lee el contenido de un archivo de disco, lo añade a la respuesta y se la envía al cliente. Por su parte, render() lee una plantilla de disco, se la pasa al motor de plantillas y el resultado devuelto lo añade a la respuesta y la envía.

La signatura de este método es la siguiente:

```
render(template)
render(template, context)
render(template, callback)
render(template, context, callback)
```

Parámetro	Tipo de datos	Descripción
template	string	Nombre de la plantilla a procesar, sin extensión de archivo.
context	object	Contexto de ejecución específico bajo el cual se procesará la plantilla.
callback	function	Función a invocar si se produce un error durante el procesamiento: fn(error, result).

Por lo general, el método render() se ejecuta cuando definimos las rutas. He aquí un ejemplo ilustrativo que muestra cómo solicitar el procesamiento de una plantilla con el correspondiente envío implícito de su resultado al cliente:

```
app.get("/index.html", function(req, res) {
  res.render("index", {title: "Express", max: 10});
});
```

Para que este ejemplo funcione, es necesario que en la carpeta de vistas, recordemos generalmente views, exista una plantilla con el nombre index, en el caso de Handlebars, index.hbs, según la extensión de archivo configurada.

Recordemos, el proceso mediante el cual se ejecuta una plantilla y se extrae el resultado a remitir al cliente se conoce formalmente como representación o reproducción (*rendering*).

Organización de plantillas

Las plantillas se deben ubicar en la carpeta de plantillas que, por convenio y buenas prácticas, es views. Hasta aquí todo bien. La cuestión es qué ocurre cuando un proyecto es de mediano o gran tamaño. Si la aplicación tiene cientos de plantillas, ubicarlas todas ellas juntas sin ninguna organización adicional en la carpeta de plantillas es posible, pero poco recomendable, pues dificulta su mantenimiento. En su lugar, la carpeta views se suele organizar en subcarpetas, cada una de ellas dedicada a un determinado componente de la aplicación. Así, es más fácil mantener el código.

En estos casos, cómo se indica el nombre de una plantilla en el método render(). Por ejemplo, supongamos que tenemos una carpeta bands que contiene las plantillas relacionadas con un componente que trabaja con información sobre bandas de música. Dentro de ella, disponemos de varias plantillas. Ahora, supongamos que deseamos reproducir aquella cuyo nombre de archivo es list.hbs.

En este caso, hay que tener claro qué es el nombre y qué la extensión. La extensión no se indica, sólo el nombre. En nuestro caso, usamos .hbs como extensión del motor de plantilla Handlebars. Por otra parte, hay que tener claro cómo separar la subcarpeta de la plantilla. Para ello, se usa ni más ni menos que la diagonal (/). Así pues, la plantilla bands/list.hbs ubicada en el directorio views se indicará de manera tan sencilla como bands/list:

```
res.render("bands/list", {…});
```

Contexto de ejecución

Cada vez que invocamos el motor de plantillas para reproducir una plantilla y, así, obtener la respuesta a remitir al cliente, se crea un contexto de ejecución (*execution context*), o simplemente contexto (*context*), esto es, un conjunto de datos específicos de esa reproducción en particular a los que puede acceder el motor para personalizar el resultado. Cada uno de estos datos se conoce formalmente como variable local (*local variable*), porque se acceden como variables en las plantillas.

Por ejemplo, si tenemos:

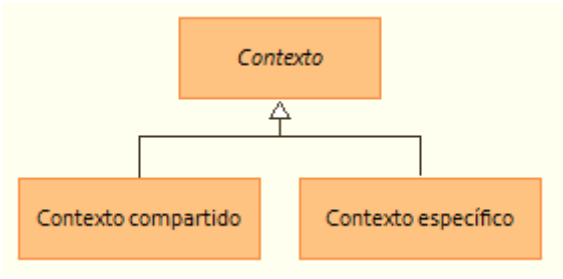
```
res.render("index", {title: "Express", max: 10});
```

En una plantilla Handlebars, el acceso a title es tan sencillo como:

```
{{title}}
```

En una expresión de Handlebars, se accede al dato usando el nombre de su propiedad en el contexto. Como si title se hubiera definido en la propia plantilla, pero haciéndose a nivel de contexto.

Existe dos contextos de ejecución, el específico y el compartido.



El contexto compartido (*shared context*) es aquel que contiene datos accesibles en *todas* las reproducciones, independientemente de qué plantilla se esté reproduciendo o representando. El contexto específico (*specific context*) es aquel que contiene datos específicos de una determinada reproducción.

Contexto compartido

Tal como acabamos de introducir, el contexto compartido (*shared context*) es aquel que contiene datos accesibles en *todas* las reproducciones. Se configura mediante la propiedad locals de la aplicación. Esta propiedad es un objeto que contiene las variables locales que se pueden utilizar en las expresiones o sentencias en línea de *todas* las plantillas de la aplicación.

Por ejemplo, para definir las variables locales compartidas appName y version, usaremos:

```
app.locals.appName = "testApp";
app.locals.version = "1.0.0";
```


Contexto específico

El contexto específico (*specific context*) es aquel que contiene variables locales específicas para una determinada reproducción. Se indica mediante el parámetro context del método render().

Por ejemplo, para proporcionar la variable local específica title, se usará:

```
res.render("index", {title: "Un título para la página"});
```

Cuando se define una variable local con el mismo nombre tanto en el contexto específico de la reproducción como en el compartido, se produce lo que se conoce formalmente como colisión de variables locales (*local variable collision*). El motor de plantillas deshace el conflicto dando prioridad a las definiciones específicas sobre las compartidas.

Ámbito

El ámbito (*scope*) es el objeto del contexto que contiene el modelo de datos que debe utilizar la plantilla. Por convenio y buenas prácticas, se pasa a través de la propiedad scope del contexto específico; no tiene sentido hacerlo a través del compartido.

Por buenas prácticas, es importante que la plantilla *no* acceda directamente al modelo de datos. Cualquier entidad de datos almacenada, por ejemplo, en una base de datos, se debe pasar a render() mediante una propiedad scope. La plantilla debe recibir todos sus datos de su contexto y, a partir de ahí, procesar la plantilla. No debe de acceder a una base de datos o servicio REST directamente. Es importante tener claro que los datos almacenados en scope se pueden pasar directamente a través del contexto. Pero por convenio y buenas prácticas, en el contexto se separa aquellos datos que representan una entidad, por ejemplo, una fila o un documento de una base de datos, del resto de propiedades. Dejando claro así que esos son datos de entidad.

He aquí un ejemplo ilustrativo:

```
var cust = {id: "550e8400-e29b-41d4-a716-44655440000", name: "Google, Inc."};
res.render("customer", {title: "Customer data", scope: cust});
```

A continuación, un ejemplo de su uso en una plantilla Handlebars:

```
<!DOCTYPE html>

<html>
  <head>
    <title>{{title}}</title>
  </head>
  <body>
    Client id.: {{scope.id}}<br>
    Client name: {{scope.name}}
  </body>
</html>
```

Configuración de los motores de plantillas

Ya hemos visto cómo configurar la carpeta de plantillas y cómo solicitar a la aplicación Express que procese una determinada plantilla con un contexto específico. Ha llegado el momento de configurar qué motores de plantillas se usan en la aplicación.

Como Express puede trabajar con varios motores de plantillas, es necesario que podamos diferenciar a qué motor pertenece cada plantilla. Para ello, se utiliza la extensión del archivo plantilla. Una vez tenemos clara la extensión a asociar a cada motor de plantilla, lo siguiente es decirle a Express: *las plantillas con esta extensión pásaselas a esta función, la cual actúa como su motor de plantillas*.

Configuración del motor de plantillas principal

El proceso para configurar el motor de plantillas principal de la aplicación es muy sencillo:

1. Configurar la extensión de las plantillas asociadas al motor principal mediante la opción view engine.
2. Configurar el motor de plantillas que procesará las plantillas indicadas en la opción view engine mediante el método engine().

Opción view engine

Mediante la opción view engine, indicamos la extensión de las plantillas del motor principal:

```
app.set("view engine", ext)
```

Parámetro	Tipo de datos	Descripción
ext	string	Extensión de las plantillas, sin punto.

He aquí unos ejemplos ilustrativos:

```
//configuración de EJS como predeterminado
app.set("view engine", "ejs");

//configuración de Handlebars como predeterminado
app.set("view engine", "hbs");
```

Método engine()

Ya sabemos cómo indicar la extensión principal de las plantillas. Ahora, vamos a indicar qué motor debe procesar las plantillas con esa extensión. Para este fin, se utiliza el método engine() de la aplicación:

```
app.engine(ext, callback)
```

Parámetro	Tipo de datos	Descripción
ext	string	Extensión de los archivos que procesará el motor de plantillas.
callback	function	Función que implementa y actúa como motor de plantillas.

Por ejemplo, he aquí dos ejemplos de asociación de extensión de plantillas a su correspondiente motor:

```
//EJS
app.engine("ejs", require("ejs").renderFile);

//Handlebars
app.engine("hbs", require("hbs").__express);
```

Como puede observar, cada motor dispone de su propia función, la cual hay que comunicar a la aplicación para que así pueda pasar las plantillas con la extensión indicada a la función que implementa el motor.

Realmente, los motores de plantillas EJS y Handlebars no requieren comunicar el motor de plantillas mediante engine(), siempre que usemos las extensiones por convenio, ejs y hbs, respectivamente. Pero es buena práctica hacerlo.