Formularios HTML

Tiempo estimado: 10min

El envío de datos por parte del usuario a la aplicación web es una parte indispensable hoy en día. Para poder enviar datos a los servidores se utiliza los formularios HTML. A partir de estos formularios, se puede enviar el identificador y las credenciales de autenticación, información de un nuevo empleado, archivos, etc.

La lección comienza recordando el concepto de formulario HTML y los aspectos a tener en cuenta para el procesamiento de sus datos en el servidor. A continuación, se muestra los dos métodos para enviar los datos de un formulario a la aplicación Express, mediante los métodos GET y POST. Para finalizar, se presenta el componente de *middleware* body-parser que analiza las peticiones HTTP y, si hay datos en el cuerpo, los pone a disposición de la aplicación mediante la propiedad req.body.

Al finalizar la lección, el estudiante sabrá:

- Cuáles son los atributos de los formularios a tener en cuenta para enviar datos al servidor.
- Cuándo usar GET y POST para enviar datos de un formulario.
- Cómo procesar el cuerpo de las peticiones HTTP mediante body-parser.

Introducción

Un formulario (*form*) es un elemento HTML que permite introducir datos al usuario para su envío a la aplicación web. Son uno de los principales puntos de interacción entre el usuario y la aplicación. Por ejemplo, a través de un formulario se puede proporcionar las credenciales de una cuenta de usuario, los datos de un pedido, de un empleado, etc.

En una página web, se representa un formulario mediante un elemento <form>, el cual delimita los elementos a través de los cuales el usuario proporciona datos, conocidos formalmente como campos (*fields*). Además, contiene botones que puede usar el usuario para enviar los datos introducidos al servidor web, para vaciar el formulario, etc.

Elemento <form>

El elemento <form> representa un formulario y actúa a modo de delimitador de los elementos de entrada de datos. Mediante el atributo method se indica el método HTTP usado por el navegador para remitir los datos al servidor. Los dos posibles valores son post y get. Hay que decir que el método también se puede indicar en el botón de envío del formulario, mediante su atributo formmethod.

Atendiendo al método HTTP elegido para enviar los datos, éstos se transmitirán en el cuerpo del mensaje o bien en la cadena de consulta de la URL. Cuando se usa el método POST, los datos se envían en el cuerpo de la petición HTTP. En este caso, podemos indicarle al servidor el tipo de contenido mediante el atributo enctype del formulario o del atributo formenctype del botón de envío.

La URL a la que remitir el formulario se indica mediante el atributo action. El cual se puede sobrescribir mediante el atributo formaction de los botones de envío. Cuando no se indica ninguna URL, se remitirá al mismo recurso que contiene el formulario.

Veamos un ejemplo ilustrativo:

```
<form action="/form/login" method="post">
  <label>Username: <input type="text" name="username"></label>
  <label>Password: <input type="password" name="password"></label>
  <button type="submit">Log in"</button>
</form>
```


Envío de datos mediante GET

Recordemos que el método GET es un método que se utiliza principalmente para solicitar cosas del servidor. Sin que afecte a su contenido. Este método no se suele utilizar para remitir datos al servidor para su almacenamiento, modificación o borrado. Se debe de utilizar sólo para pedirle datos al servidor.

También tenemos que recordar que cuando se envía una petición GET, el mensaje no puede contener cuerpo. Por lo que los datos del formulario se deberán especificar en la URL del recurso, mediante la cadena de consulta. Por ejemplo, si remitimos el formulario del ejemplo anterior mediante GET, la petición HTTP remitida por el navegador a la aplicación se asemejará a algo como:

```
GET /form/login?username=elvis&password=costello HTTP/1.1
Host: mi.com
```

Es importante tener claro que la URL final, la que contiene el recurso indicado en el atributo action o formaction y los campos del formulario, la genera el navegador cuando redacte la petición. Así pues, el formulario anterior mediante GET será como sigue:

```
<form action="/form/login" method="get">
  <label>Username: <input type="text" name="username"></label>
  <label>Password: <input type="password" name="password"></label>
  <button type="submit">Log in"</button>
</form>
```

Envío de datos mediante POST

Generalmente, cuando un usuario tiene que enviar datos al servidor para su almacenamiento o procesamiento, se suele utilizar el método POST, en vez de GET. En este caso, los datos proporcionados por el usuario en el formulario se adjuntan en el cuerpo de la petición HTTP. Como el mensaje remitido por el cliente contiene cuerpo, éste debe indicar el tipo de su contenido que, recordemos, se indica mediante el atributo enctype del formulario o formenctype del botón de envío. Los posibles valores de estos atributos son application/x-www-form-urlencoded, multipart/form-data y text/plain, siendo application/x-www-form-urlencoded el valor predeterminado cuando no se indica nada explícitamente.

He aquí el ejemplo de petición HTTP para el formulario de ejemplo remitido mediante POST:

```
POST /form/login HTTP/1.1
Host: mi.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

username=elvis&password=costello
```

Cuando se usa application/x-www-form-urlencoded, el navegador adjunta los campos del formulario mediante pares nombre=valor, separándolos por el signo &. Es la misma sintaxis de la cadena de consulta de las URLs.

Envío de archivos

Cuando el usuario tiene que adjuntar archivos de su disco duro al servidor, hay que utilizar necesariamente el método POST y el tipo de contenido multipart/form-data.

Middleware body-parser

Para facilitar el acceso a los datos del cuerpo remitidos en las peticiones HTTP, Express proporciona el componente body-parser. Esta pieza de *middleware* lo que hace es analizar el cuerpo remitido y ponerlo a disposición de la aplicación mediante la propiedad body del objeto req. body-parser proporciona varias funciones cada una de ellas con el objeto de procesar un determinado tipo de contenido:

- json() para contenido formateado en JSON.
- text() para contenido de tipo texto.
- urlencoded() para contenido de tipo application/x-www-form-urlencoded.
- raw() para contenido como una secuencia de *bytes*.

Actualmente, no proporciona ninguna función para procesar cuerpos multiparte, recordemos, aquellos que

tienen como tipo de contenido multipart/form-data.

Atendiendo a los tipos de contenido que debe procesar la aplicación, registraremos cero, una o más funciones de *middleware*, una para cada uno de los tipos que puede procesar.

body-parser.json()

La función json() devuelve una función de *middleware* para procesar cuerpos de tipo JSON mediante la función JSON.parse():

```
function json() : function
function json(options) : function
```

Parámetro	Tipo de datos	Descripción
options	object	Opciones del *parsing*: • type (string). Tipo de contenido que puede procesar. Se puede usar comodines como *. Valor predeterminado: application/json. • strict (boolean). ¿Sólo *arrays* y objetos? En otro caso, cualquier valor analizable por JSON.parse(). Valor predeterminado: true. • limit (string o number). Tamaño máximo del cuerpo. Si es un número, el valor será en *bytes*. Si es una cadena de texto, se puede indicar la unidad de medida: b, kb, mb, gb... Valor predeterminado: 100kb. • reviver (object). Segundo argumento a pasar a la función JSON.parse().

Esta función lo que hace es comprobar si el tipo de contenido es el indicado y, si es así, coger el cuerpo del mensaje, pasarlo por la función JSON.parse() y asignar el valor devuelto a la propiedad body de req.

He aquí algunos ejemplos ilustrativos:

```
app.use(bodyParser.json());
app.use(bodyParser.json({type: "application/*+json", limit: "512b"});
```

body-parser.urlencoded()

Mediante urlencoded(), se obtiene una función de *middleware* para analizar cuerpos cuyo tipo de contenido es application/x-www-form-urlencoded:

```
function urlencoded() : function
function urlencoded(options) : function
```

Parámetro	Tipo de datos	Descripción
options	object	Opciones del *parsing*: • type (string). Tipo de contenido que puede procesar. Valor predeterminado: application/x-www-form-urlencoded. • limit (string o number). Tamaño máximo del cuerpo. Valor predeterminado: 100kb. • parameterLimit (number). Número máximo de parámetros a analizar. Valor predeterminado: 1000. • extended (boolean). ¿Analizar el cuerpo mediante el módulo qs o, por el contrario, mediante querystring? Se recomienda false, o sea, querystring. Valor predeterminado: true, es decir, qs.

Ejemplos:

```
app.use(bodyParser.urlencoded());
app.use(bodyParser.urlencoded({extended: false, limit: "512b"});
```

body-parser.text()

Cuando el cuerpo puede contener datos de tipo text como, por ejemplo, text/plain o text/html, se puede utilizar text() para obtener una función de *middleware* que analice el contenido y lo ponga disponible mediante la propiedad body de la petición HTTP:

```
function text() : function
function text(options) : function
```

Parámetro	Tipo de datos	Descripción
options	object	Opciones del *parsing*: • type (string). Tipo de contenido que puede procesar. Valor predeterminado: text/plain. • limit (string o number). Tamaño máximo del cuerpo. Valor predeterminado: 100kb. • defaultCharset (string). Conjunto de caracteres predeterminado si la petición no indica ninguno explícitamente en el campo de cabecera Content-Type. Valor predeterminado: utf-8.

Veamos un ejemplo:

```
app.use(bodyParser.text({type: "text/html", limit: "512b"});
```

body-parser.raw()

Cuando el contenido es un flujo de *bytes*, podemos utilizar raw() para obtener una función de *middleware* que analice este tipo de contenido:

```
function raw() : function
function raw(options) : function
```

Parámetro	Tipo de datos	Descripción
options	object	Opciones del *parsing*: • type (string). Tipo de contenido que puede procesar. Valor predeterminado: application/octet-stream. • limit (string o number). Tamaño máximo del cuerpo. Valor predeterminado: 100kb.

Registro de funciones de middleware

Hasta el momento, el 99% de los ejemplos de registro de funciones de *middleware* en la pila de procesamiento que hemos presentado se ejecutaban con cada petición HTTP recibida. Usaban la siguiente sintaxis de la función use():

```
use(middleware)
```

Pero también es posible restringir la ejecución de las funciones a sólo determinadas rutas. Esto se puede hacer de dos maneras. La primera, mediante use():

```
use(path, middleware)
```

La otra opción es mediante los métodos get(), post(), put(), delete() y familia. Cuando presentamos sus signaturas, lo hicimos como sigue:

```
método(path, handler)
```

Pero siendo sinceros, existe una sobrecarga que permite pasar varios controladores de petición y, por ende, incluso funciones de *middleware* que son, en efecto, controladores de petición:

```
método(path, handler1, handler2, handler3…)
```

Así pues, si necesitamos un determinado analizador de cuerpos para una ruta concreta, he aquí unos ejemplos ilustrativos de cómo hacerlo:

```
app.use("/form/login", bodyParser.urlencoded());

app.post("/form/login", bodyParser.urlencoded(), function(req, res) {
  ...
});
```