3. Desarrollo de servicios con Node.js

Anexo III. Cargar imágenes y autenticación de usuario con Node

Programación de Servicios y Procesos

Arturo Bernal
Nacho Iborra
Javier Carrasco

Tabla de contenido

1. Imágenes y *base64*

Si quieres subir algún archivo a un servidor web de forma tradicional, debes añadir un elemento *<input type = "file"…* en un formulario y enviarlo como *multipart -form-data*. Con este tipo de envío de archivos, puedes simplemente obtener el archivo en servidores web basados en PHP o JSP, y colocar el archivo en la carpeta deseada.

También puedes tratar estos envíos en Node a través de algunos módulos de terceros útiles, pero en este anexo centraremos la atención en una forma específica de subir imágenes a un servidor, convirtiéndolas en atributos codificados JSON.

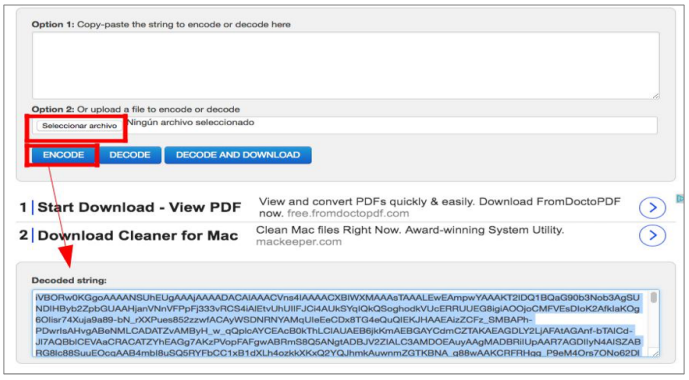
Quizás te estés preguntando… ¿cómo diablos se puede poner una imagen dentro de una cadena? La respuesta a esta pregunta es realmente simple, de hecho: se pueden codificar los bytes de la imagen y enviar estos bytes codificados en una cadena dentro de un objeto JSON. Esta tarea la realiza el cliente, por lo que centraremos en ella en la siguiente unidad, pero por ahora, se verá cómo obtener esta imagen del servidor Node una vez que ha sido enviada por el cliente.

En primer lugar, necesitas saber que las imágenes están codificadas en JSON usando codificación *base64*. Así que solo tienes que obtener el campo de imagen del objeto JSON y decodificarlo desde *base64* en sus bytes originales. Luego, podrás guardar esos bytes en un archivo a través del módulo *fs*.

Supón que obtienes del cliente este objeto, que contiene el nombre del archivo de imagen y los bytes de la imagen (codificados en *base64*), en dos atributos separados del mismo objeto:

```
let file = {
    fileName: 'image.png',
    bytes: 'iVBORw0KGgoAAAA.....'
};
```

Puedes descargar el código completo de este objeto en los recursos de esta sesión, junto a este anexo. También puedes probar con cualquier imagen en webs como esta[1]. Simplemente carga una imagen, haz clic en el botón ENCODE y copia la codificación *base64* (resaltada en azul en la imagen a continuación) en tu código, reemplazando el atributo bytes del objeto anterior.



1 FREEFORMATTER.COM (https://www.freeformatter.com/base64-encoder.html#ad-output)

Lo que se hará con la imagen, una vez obtenidos los bytes del objeto Javascript, es decodificarlos y almacenarlos en la carpeta deseada con el nombre del archivo original. El código para hacer esto sería más o menos así:

```
const fs = require('fs');

let file = { /* ... same information shown above ... */ };
let data = Buffer.from(file.bytes, 'base64');
fs.writeFileSync(file.fileName, data);
```

Ejercicio 1

Cree un proyecto llamado ***"Exercise_ImageUpload"*** en tu espacio de trabajo y copia/adapta el código anterior en un archivo fuente llamado *"image_upload.js"*. Elige y codifica una imagen, y luego pega los bytes correspondientes en el objeto *file* del ejemplo. Luego, intenta ejecutar la aplicación y observa cómo se crea un nuevo archivo de imagen dentro de la carpeta de la aplicación de Node. Intenta abrirlo y comprueba si es tu imagen elegida.

2. Uso de la autenticación mediante token

Existen varias formas de autenticar a un usuario. La más tradicional es utilizar sesiones para almacenar las credenciales de usuario (normalmente su identificación o inicio de sesión), una vez que el servidor ha verificado que el usuario y la contraseña proporcionados por el lado del cliente son correctos. Sin embargo, este tipo de gestión de usuarios no siempre es posible, ya que muchas aplicaciones cliente, como las aplicaciones de escritorio o híbridas, no dependen de las sesiones para almacenar información.

Entonces, lo que se verá aquí es una forma más flexible y universal de autenticar a los usuarios y almacenar sus datos una vez que hayan iniciado sesión. Hablamos de autenticación mediante token.

2.1. Fundamentos de la autenticación de tokens

Un token es una cadena que almacena información sobre un usuario. Esta cadena está codificada para ser transferida entre clientes y servidores y tiene una firma secreta para que solo un servidor pueda determinar si un token es válido o no. Los pasos básicos de este proceso son:

1. El cliente envía sus datos de autenticación (nombre de usuario y contraseña) al servidor.
2. El servidor toma esta información para verificar si hay un usuario con estas credenciales en la base de datos. Si es así, genera un token, una cadena codificada con información útil sobre el usuario, como su identificación o inicio de sesión. Luego, firma el token con una palabra secreta especial para que solo este servidor pueda decodificar el token cuando regrese del cliente. Finalmente, el servidor envía el token al cliente.

3. El cliente recibe este token y lo usa para cada comunicación futura con el servidor, de modo que, cada vez que el servidor recibe una solicitud de ese cliente, puede verificar si el cliente ya ha sido validado o no.

Los tokens suelen tener un tiempo de caducidad limitado N (pueden ser solo unos minutos, u horas, o días, semanas, …, dependiendo de la aplicación que se este desarrollando). Pero, cada vez que un cliente envía un token válido al servidor con una nueva solicitud, este token puede (debe) renovarse, añadiendo N nuevos minutos/segundos/horas… Sin embargo, si el cliente pasa más tiempo sin enviar solicitudes al servidor, su token caducará y deberá volver a iniciar sesión.

Se utilizará un estándar llamado JWT (que significa JSON Web Token) para crear y validar tokens entre clientes y servidores, utilizando objetos JSON.

2.1.1. Estructura de un token JSON

Aunque no se explorará esta información en detalle, un token consiste en una cadena dividida en tres componentes: un encabezado (con información sobre el algoritmo de codificación, tipo de token, …), una carga útil (con información almacenada en el token, como información del usuario, tiempo de vencimiento, …) y una firma (una clave secreta utilizada por el servidor para validar el token cuando se decodifica).

Toda esta información se codifica utilizando *base64* y se envía entre el cliente y el servidor con cada proceso de solicitud/respuesta (*request/response*). Aquí puedes ver la apariencia de uno de estos tokens:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtZXNzYWdlIjoiSldUIFJ1bGVzI
SIsImlhdCI6MTQ1OTQ0ODExOSwiZXhwIjoxNDU5NDU0NTE5fQ.-yIVBD5b73C75osbmwwshQNRC7frWUYrqaTjTpza2y4
```

2.1.2. ¿Por qué utilizar la autenticación mediante token?

El mecanismo de autenticación de tokens tiene algunas ventajas si se compara con el mecanismo de sesión tradicional:

- Los tokens no tienen estado, es decir, el servidor no tiene que almacenar ninguna información sobre los tokens que produce o recibe. Toda la información está contenida en el token.
- Es compatible con cualquier aplicación cliente-servidor (web, móvil, escritorio, híbrida, …).
- También se puede almacenar información adicional en los tokens, de modo que se pueda leer esta información cada vez que se descodifique el token.

3. Autenticación con Node

Observa una forma sencilla de implementar la autenticación basada en token con Node. Suón que el cliente envía sus credenciales en un objeto JSON con dos atributos (llamados *login* y *password*).

Para empezar, necesitas instalar un módulo de terceros en el proyecto Node. Este módulo se llama ***"jsonwebtoken"***, y está disponible en el repositorio oficial de *npm*, por lo que solo necesitas teclear los comandos habituales (*npm install* y, previamente, *npm init* si aún no tienes el archivo *"package.json"*). Después, ya puedes incluir este módulo en el archivo fuente:

```
const jwt = require('jsonwebtoken');
```

3.1. Generar un token

Una vez comprobado que las credenciales de usuario son correctas, es necesario generar un token. Para hacer esto, se utiliza el método ***sign*** del módulo *jsonwebtoken*. Tiene tres parámetros:

- La información que se añadirá al token (como un objeto Javascript). Por ejemplo, se puede añadir el inicio de sesión del usuario.
- La palabra secreta con la que firmar el token. Esta palabra secreta suele ser la misma para todos los tokens firmados por el mismo servidor Node, por lo que puede almacenarse en una constante, o en un archivo separado desde el que se pueda leer.
- Opciones adicionales, como el tiempo de vencimiento. Este tiempo se puede expresar en muchos formatos, como *"2 days"*, *"3h"*, *"15d"*, *"5 minutes"*, …

La siguiente función genera un token con el inicio de sesión del usuario, con un tiempo de vencimiento de 2 horas, y usando la palabra secreta especificada para codificarlo.

```
const secretWord = "DAMsecret";

let generateToken = login => {
    let token = jwt.sign({login: login}, secretWord,
                         {expiresIn:"30 minutes"});
    return token;
}
```

Puede utilizarse dentro de una aplicación *Express*, llamando desde un servicio llamado *login*, para generar el token una vez que el usuario haya sido validado correctamente:

```
app.post('/login', (req, res) => {

    // Get user credentials from the request

    let userClient = {
        login: req.body.login,
        password: req.body.password
    };
```

```
    // Look for user in the collection

    User.findOne({login: userClient.login, password: userClient.password}).then(data => {
        // User is valid. Generate token
        if (data != null) {
            let token = generateToken(userClient.login);
            let result = {error: false, token: token};
            res.send(result);
        // User not found, generate error message
        } else {
            let result = {error: true, errorMessage: "Invalid user"};
            res.send(result);
        }
    }).catch (error => {

        // Error searching user. Generate error message
        let result = {error: true, errorMessage: "Error trying to validate user"};
        res.send(result);
    });
});
```

3.2. Validar y renovar un token existente

Si se obtiene un token de una solicitud de cliente, generalmente se envía a través de un encabezado de solicitud llamado *"Authorization"*. Debe leerse este encabezado y verificar el token. Opcionalmente (pero habitualmente) también se renueva. Para comprobar si un token es válido, se puede usar el método *verify()* del módulo *jsonwebtoken*. Obtiene dos parámetros: el token a validar y la palabra secreta para validarlo. Por lo que, un método para verificar un token podría ser como este:

```
let validateToken = token => {
    try {
        let result = jwt.verify(token, secretWord);
        return result;
    } catch (e) {}
}
```

Como puedes ver, llamando al método *verify()* se puede lanzar una excepción si el token no es válido, por lo que debe utilizarse un *try-catch*. Si la validación tiene éxito, se podrá acceder a toda la información incluida en el token, como el inicio de sesión del usuario:

```
let result = jwt.verify(token, secretWord);
console.log(result.login);
```

También se puede usar el método *generateToken()* mostrado anteriormente para renovar el token una vez comprobado que es válido.

```
let result = validateToken(someToken);

if (result) {
    let newToken = generateToken(result.login);
    … // send this new token to the client instead of the old one
}
```

3.3. Enviar el token entre cliente y servidor

Una vez que un usuario es validado y el servidor le envía un nuevo token, este token se envía entre el cliente y el servidor para cada solicitud del cliente.

- Siempre que el servidor envía un token a un cliente, generalmente se envía en la respuesta, como un atributo JSON.

```
let result = {… token: token};
res.send(result);
```

- Si el cliente desea devolver el token al servidor, debe utilizar el encabezado *"Authorization"*. Entonces, dentro del código del servidor, deberá accederse a este encabezado para obtener el token y validarlo:

```
let token = req.headers['authorization'];
let result = validateToken(token);
if (token) …
```

  En este caso, el servidor puede renovar el token llamando al método *generateToken()*, como se ha visto en el ejemplo anterior.

- Si el token no es válido, se deberá enviar un mensaje de error al cliente en lugar de la respuesta deseada.

> IMPORTANTE: de acuerdo con los estándares, el token debe enviarse a través del encabezado *"Authorization"* con un prefijo *"Bearer"*, por lo que el contenido real de este encabezado debe ser *"Bearer… token…"* y, en el lado del servidor, deberá omitirse este prefijo al leer el token. Sin embargo, se omitirá este prefijo en algunos de los ejemplos y ejercicios explicados en esta unidad. Pero debes ser consciente de ello.

Ejercicio 2

Se creará un ejemplo simple de cómo validar a un usuario y requerir un token para acceder a algunos servicios restringidos. Crea una carpeta llamada ***"Exercise_SimpleToken"*** en tu espacio de trabajo y sigue los siguientes pasos:

- Crea un archivo *"package.json"* ejecutando el comando *npm init*.
- Instala los módulos *express*, *mongoose*, *jsonwebtoken* y *body-parser* mediante del comando *npm install*.
- Crea un archivo fuente llamado *"index.js"* y añádele el siguiente código:

```
const express = require('express');
const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const secretWord = "DAMsecret";
```

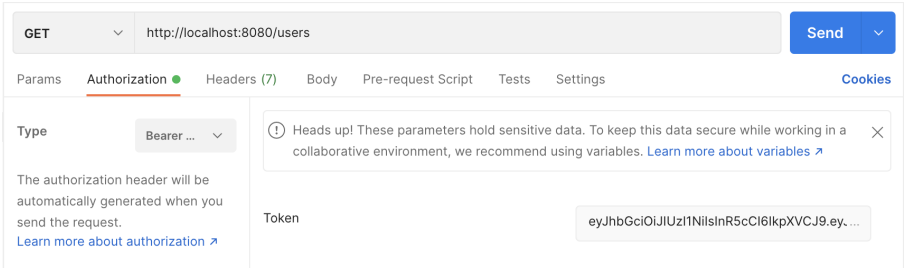
```
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost:27017/users');

let userSchema = new mongoose.Schema({
    login: {
        type: String,
        required: true,
        minlength: 1,
        unique: true
    },
    password: {
        type: String,
        required: true,
        minlength: 4
    },
    name: {
        type: String,
        required: true,
        minlength: 1
    }
});

let User = mongoose.model('User', userSchema);
```

Ten en cuenta que el campo *login* se ha configurado como único, es decir, no puede haber dos usuarios diferentes con el mismo inicio de sesión.

- Después, añade el código para crear un servidor *Express* para responder a los siguientes servicios:
    - ***/login*** a través del comando POST: se obtendrán los campos *login* y *password* en el cuerpo de la solicitud, valida si existe un usuario con estas credenciales en la colección de usuarios (usando el modelo de usuario que se muestra arriba), y luego genera un token durante 30 minutos, con la palabra secreta que quieras. Devolverás el token generado junto con un campo *error*, como de costumbre.
    - ***/users*** a través del comando POST: añadirá un nuevo usuario. Obtendrás los datos del usuario (*name*, *login* y *password*) del cuerpo de la solicitud y añádelos con el modelo de usuario. Enviará un *error* junto con el nuevo usuario agregado.
    - ***/users*** a través del comando GET. Este servicio tiene acceso restringido, solo puede ser llamado con un token válido. Por lo tanto, el servidor debe leer el token del encabezado *"Authorization"* de la solicitud y, si es válido, devolverá la lista de usuarios de la colección de *users* como respuesta.

Una vez que el usuario ha sido validado a través del servicio */login*, el token generado debe enviarse en cada nueva solicitud (aparte de */login*), y debe ser renovado por el servidor y devuelto al cliente.

Puedes probar todos estos servicios con *Postman*. Para enviar algo en un encabezado de solicitud, puedes hacer clic en la pestaña *Authorization* debajo de la URL y especificar el tipo (*Bearer* token), y después pegar el token en el formulario del lado derecho:



Ten en cuenta que estás enviando un token *Bearer* de esta manera, es decir, habrá un prefijo *“Bearer”* en el encabezado del token que deberás omitir en el servidor (puedes usar el método de *substring()* para cortar el carácter en la séptima posición).