Clase 18: Material Complementario

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:35
Libro:	Clase 18: Material Complementario		

Tabla de contenidos

1. Json Web Token

Json Web Token

1.1. Json Web Token

Analogía del guardia en el edificio

Imagina un edificio con acceso restringido, donde sólo ciertas personas tienen permiso para entrar. Aquí está cómo se relacionan el token y la clave secreta con esta analogía:

1. **El Edificio**:
    - Representa tu aplicación o servicio. Solo los usuarios autorizados pueden acceder a ella.

2. **El Guardia de Seguridad**:
    - El guardia representa el servidor que controla el acceso. Su tarea es verificar la identidad de las personas que intentan entrar al edificio.

3. **La Identificación (Token)**:
    - Cuando una persona (usuario) entra al edificio, el guardia le proporciona una identificación especial (token) que confirma que ha sido verificada y tiene permiso para estar dentro. Este token contiene información sobre el usuario, como su nombre y el tiempo que estará autorizado para estar dentro.
        - **Ejemplo**: El token puede incluir un nombre de usuario y una fecha de expiración, como "Juan Pérez - acceso válido hasta las 2:00 PM".

4. **La Clave Secreta**:
    - La clave secreta es como el procedimiento secreto que el guardia utiliza para asegurarse de que solo él puede emitir identificaciones válidas. Si alguien intentara falsificar una identificación, el guardia notaría que la identificación no se ajusta a los procedimientos que él sigue (la firma).
        - **Ejemplo**: Solo el guardia tiene acceso a un libro de registro que contiene las reglas de quién puede entrar y cómo se debe ver una identificación válida. Si alguien intenta entrar con una identificación que no fue emitida por el guardia, no podrá pasar.

5. **Verificación de la Identificación**:
    - Cuando una persona intenta volver a entrar al edificio, el guardia verifica su identificación (token). Si la identificación fue emitida por él (usando la clave secreta) y aún es válida (no ha expirado), el guardia le permitirá la entrada.
        - **Ejemplo**: Si la identificación de Juan está en regla y no ha expirado, el guardia le permitirá el acceso. Si la identificación está manipulada o ha caducado, no podrá entrar.

6. **Acceso a Áreas Restringidas**:
    - Dentro del edificio, hay áreas a las que solo algunas personas pueden acceder. La identificación (token) puede incluir información sobre a qué áreas tiene acceso el usuario. El guardia puede verificar esto antes de permitirle entrar a esas áreas.
        - **Ejemplo**: Si Juan tiene permiso solo para el primer piso, pero intenta acceder al segundo, el guardia lo detendrá y le recordará sus limitaciones.

2. Sesiones y middleware

Temario:

- Sesiones.
- Middleware.

2.1. Sesiones y middleware

Sesiones

Las sesiones son un método para hacer que algunas variables estén disponibles en múltiples controladores sin tener que pasarlas como parámetro. A diferencia de las cookies, las variables de sesión se almacenan en el servidor y tienen un tiempo limitado de existencia.

Para identificar al usuario que generó las variables de sesión, el servidor genera una clave única que es enviada al navegador y almacenada en una cookie. Luego, cada vez que el navegador solicita otra página al mismo sitio, envía esta cookie (clave única) con la cual el servidor identifica de qué navegador proviene la petición y puede rescatar de un archivo de texto las variables de sesión que se han creado.

Después de un tiempo (configurable) sin peticiones por parte de un cliente (navegador) las variables de sesión son eliminadas automáticamente. Una variable de sesión es más segura que una cookie ya que se almacena en el servidor. Otra ventaja es que no tiene que estar enviándose continuamente como sucede con las cookies.

Uno de los usos más comunes de las variables de sesión en las aplicaciones web es el proceso de autenticación y autorización de las distintas peticiones que el usuario realiza.

En primer lugar, debemos de conocer brevemente los términos de autenticación y autorización.

- **Autenticación** es el proceso de verificar si el usuario es el mismo que él declara ser.
- **Autorización** es el proceso de determinar si un usuario tiene los privilegios para acceder a un cierto recurso al que ha solicitado acceder.

Para poder utilizar variables de sesión debemos instalar el módulo express-session en nuestro proyecto e inicializarlo correctamente.

Para instalarlo bastará con escribir npm i express-session en nuestra consola. Para configurarlo primero debemos importar el módulo e inicializarlo.

A continuación vemos un pequeño fragmento de código Node.js en el que se ilustra de una manera muy simple el proceso de autenticación y autorización mediante sesiones de express.js. Como era de esperar, hay un punto de inicio de sesión, un punto de cierre de sesión. Para ver la página que está posteada debemos autenticarnos previamente, de esta forma nuestra identidad será verificada y guardada durante la sesión. Cuando cerremos la sesión lo que se producirá internamente es un borrado de nuestra identidad en dicha sesión.

Para poner en funcionamiento debemos instalar el módulo de express-session vía npm.

npm i express-session

Vamos al archivo de configuración (app.js)

```
const session = require('express-session');

app.use(session({
    secret: 'inserte clave aqui',
    resave: false,
    saveUninitialized: true
}));
```

En este caso inicializamos el middleware de sesiones pasando como parámetro un objeto con las propiedades secret, resave y saveUninitialized. El único parámetro requerido de estos es secret, que será la cadena de texto o clave que se utilizará para generar el id de sesión que se envía en la cookie al usuario y luego verificar que el mismo no haya sido adulterado por el usuario. Este valor no debe ser sencillo de adivinar, por lo que se recomienda usar cadenas de texto aleatorias de por lo menos 20 caracteres.

A partir de este momento el objeto res que reciben todos nuestros controladores contará con una propiedad llamada session la cual podemos tanto leer como escribir.

En el siguiente ejemplo hacemos una verificación sencilla de la existencia de la variable de sesión nombre. En caso de existir saludamos al usuario usando este valor. En caso contrario consideramos que el usuario es desconocido.

```
app.get('/ejemplo', function(req, res) {
    if (req.session.nombre) {
        res.send('Hola ' + req.session.nombre) ;
    } else {
        res.send('Hola usuario desconocido.') ;
    }
});
```

A continuación vemos un ejemplo más completo donde verificamos también la existencia de una variable

de sesión llamada nombre y en base a su existencia mostraremos o no al usuario un formulario que le permita ingresar su nombre y que el mismo quede grabado en la sesión.

Así mismo le daremos al usuario un link que llevará al usuario a una ruta que se encarga de destruir la sesión, permitiendo iniciar el ciclo nuevamente.

```
app.get('/', function(req, res) {
  var conocido = Boolean(req.session.nombre);

  res.render('index', {
    title: 'Sesiones en Express.js',
    conocido: conocido,
    nombre: req.session.nombre
  });

});


app.post('/ingresar', function(req, res) {
  if (req.body.nombre) {
    req.session.nombre = req.body.nombre
  }
  res.redirect('/');
});

app.get('/salir', function (req, res) {
  req.session.destroy();
  res.redirect('/');
});
```

```
<h1>{{title}}</h1>
{{#if conocido}}
    <p>Hola {{nombre}} <a href="/salir">Salir</a></p>
{{/if}}

{{#unless conocido}}
    <p>No te conozco</p>
    <form action="/ingresar" method="post">
        <input type="text" name="nombre"><button>ingresar</button>
    </form>
{{/unless}}
```

Middleware

Un middleware es una función que se puede ejecutar antes o después del manejo de una ruta. Esta función tiene acceso al objeto Request, Response y la función next().

Las funciones middleware suelen ser utilizadas como mecanismo para verificar niveles de acceso antes de entrar en una ruta, manejo de errores, validación de datos, etc.

En resumen las funciones de middleware pueden realizar las siguientes tareas:

- Realizar cambios en la solicitud y los objetos de respuesta.
- Finalizar el ciclo de solicitud/respuestas.
- Invocar el siguiente middleware en la pila.

Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada. El siguiente ejemplo creamos un middleware sencillo para contar las vistas del usuario a una ruta determinada y almacenaremos dicho valor en una variable.

```
app.use(function(req, res, next) {
  // si no existe la variable de sesion vistas la
  // creamos como un objeto vacio
  if (!req.session.vistas) {
    req.session.vistas = {};
  }

  /**
   * buscamos una clave dentro session.vistas que
   * coincida con la url actual. Si no existe, la
   * inicializamos en 1. Si existe sumamos 1 al contador
   * de esa ruta
   */
  if (!req.session.vistas[req.originalUrl]) {
    req.session.vistas[req.originalUrl] = 1;
  } else {
    req.session.vistas[req.originalUrl]++;
  }

  next();
});


app.get('/pagina1', function(req, res) {
  /**
   * pasamos al template la variable vistas con el
   * numero devuelto por la clave que pedimos
   */
  res.render('pagina', {
    nombre: 'pagina1',
    vistas: req.session.vistas[req.originalUrl]
  });
});
```

Bibliografía:

- Express . Disponible desde la URL: https://expressjs.com/es/
- Handlebars . Disponible desde la URL: https://handlebarsjs.com/

3. Manejo de archivos multimedia, validaciones y módulos útiles

Objetivos

- Aprender a configurar y utilizar las librerías como Multer para gestionar la carga y almacenamiento de archivos multimedia como imágenes y vídeos.
- Implementar validaciones para garantizar la seguridad de las aplicaciones.
- Explorar y aplicar módulos útiles para funcionalidades extendidas en aplicaciones Node.js.

3.1. Manejo de archivos multimedia, validaciones y módulos útiles

Manejo de Archivos Multimedia

Los archivos multimedia, como imágenes y videos, son componentes esenciales en muchas aplicaciones modernas. Aprenderemos a manejar estos archivos usando herramientas como Formidable y Multer.

Configuración de Multer

```
const multer = require('multer'); const
storage = multer.diskStorage({
destination: function(req, file, cb) {
    cb(null, 'uploads/'); // Directorio donde se almacenarán los archivos
  },
  filename: function(req, file, cb) {
    cb(null, Date.now() + '-' + file.originalname); // Nombre del archivo
  }
});
const upload = multer({ storage: storage });

// Ruta para manejar la carga de archivos
app.post('/upload', upload.single('file'), function(req, res) {
  // Lógica para manejar el archivo subido
res.send('Archivo subido correctamente.');
});
```

Validaciones y Seguridad

Implementaremos validaciones para asegurar que los archivos cargados cumplan con los requisitos específicos antes de ser almacenados en el servidor. Esta práctica es crucial para evitar la manipulación maliciosa de datos.

Uso de Multer con validaciones

```
const multer = require('multer'); const
storage = multer.diskStorage({
destination: function(req, file, cb) {
    cb(null, 'uploads/'); // Directorio donde se almacenarán los archivos
  },
  filename: function(req, file, cb) {
    cb(null, Date.now() + '-' + file.originalname); // Nombre del archivo
  }
});
const fileFilter = (req, file, cb) => {
  if (file.mimetype === 'image/jpeg' || file.mimetype === 'image/png') {
    cb(null, true); // Aceptar el archivo
  } else {
    cb(null, false); // Rechazar el archivo
  }
};
const upload = multer({ storage: storage, fileFilter: fileFilter });

// Ruta para manejar la carga de archivos con validaciones
app.post('/upload', upload.single('file'), function(req, res) {
  // Lógica para manejar el archivo subido
res.send('Archivo subido correctamente.');
});
```

Módulos Útiles en Node.js

Exploraremos módulos como pdf-lib para la generación de archivos PDF dinámicos y dotenv para gestionar variables de entorno de manera segura en nuestras aplicaciones.

Uso de dotenv para Variables de Entorno

```
require('dotenv').config(); // Cargar variables de entorno desde .env
const apiKey = process.env.API_KEY; // Ejemplo de cómo acceder a una variable de entorno

// Configuración de una conexión a una API externa usando axios
const axios = require('axios');
axios.get('https://api.example.com/data', {   headers: {
    'Authorization': `Bearer ${apiKey}`
  }
}).then(response => {
console.log(response.data);
}).catch(error => {
  console.error('Error al hacer la solicitud:', error);
});
```

Práctica sistema de carga de archivos

Consigna

Objetivo: Implementar un sistema de carga de archivos multimedia en una aplicación Node.js usando Multer, con validaciones para asegurar que solo se acepten imágenes JPEG, JPG o PNG.

- **Configurar un proyecto Node.js básico con Express, utilizando HBS para mostrar contenido dinámico. (incluir parciales)**
- **Implementar el modelo-vista-controlador.**
- **Configurar el sistema para que permita realizar las cuatro operaciones básicas de un CRUD** (Crear, Leer, Actualizar y Eliminar) utilizando MongoDB como base de datos
- **Implementar la configuración de Multer según el ejemplo proporcionado.**
- **Crear una ruta /upload que maneje la carga de archivos y valide que solo se acepten archivos de imagen (JPEG, JPG, PNG).**
- **Mostrar un mensaje de confirmación al usuario cuando el archivo se cargue correctamente, mostrando detalles como el nombre y tipo de archivo.**
- **Incluir un archivo .env para manejar configuraciones sensibles y variables de entorno.**
- **Subir la actividad a GitHub, incluir archivo gitignore.**
- **Incluir archivo README a modo de documentación explicando la implementación y cualquier decisión de diseño.**