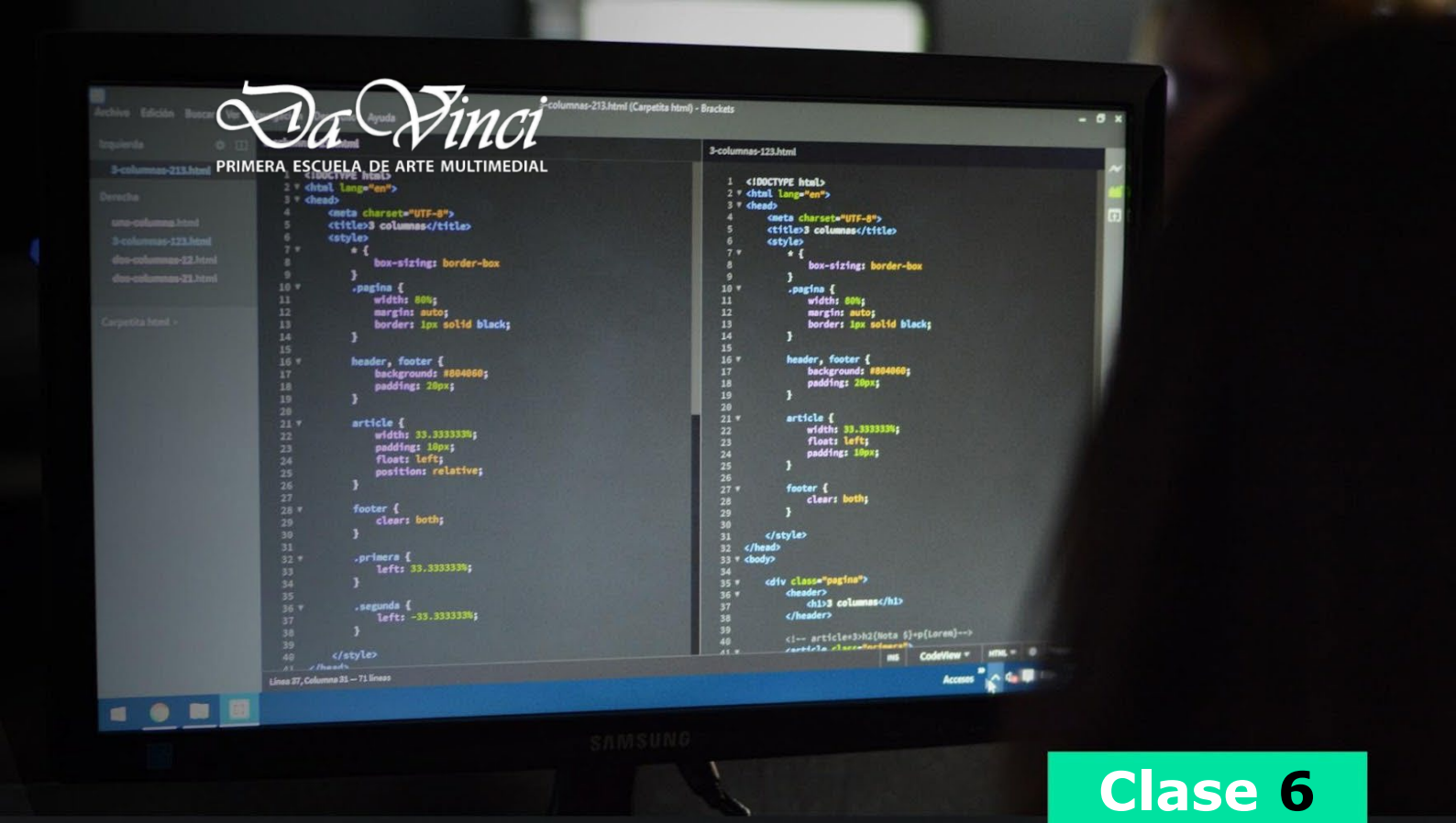

Clase 6

Diseño y Programación Web

Materia:
Aplicaciones Híbridas

Docente contenidista: MARCOS GALBÁN, Camila Belén

Middlewares

Un middleware en Express es una función que tiene acceso al objeto de solicitud (req), al objeto de respuesta (res) y a la siguiente función de middleware en el ciclo de solicitud/respuesta de la aplicación. Los middlewares pueden realizar una variedad de tareas, como ejecutar código, modificar la solicitud y la respuesta, finalizar el ciclo de solicitud/respuesta, y llamar a la siguiente función de middleware.

Partes de un Middleware en Express

El objeto de solicitud (req):

- Contiene información sobre la solicitud HTTP entrante.
- Incluye datos como los parámetros de la ruta, los datos del formulario, los encabezados HTTP, etc.

El objeto de respuesta (res):

- Utilizado para enviar una respuesta HTTP al cliente.
- Incluye métodos para configurar el estado de la respuesta, enviar datos, redirigir, etc.

La función next:

- Una función que se llama para pasar el control al siguiente middleware en la pila.
- Si no se llama a next(), la solicitud quedará colgada.

Ejemplo de middleware

```
function middlewareEjemplo(req, res, next) {
    // Ejecutar cualquier código
    console.log('Middleware ejecutado');
```

```
    // Modificar el objeto de solicitud (req) o respuesta (res)
    req.miPropiedad = 'valor';

    // Finalizar el ciclo de solicitud/respuesta
    if (req.miPropiedad === 'valor') {
       res.send('Ciclo de solicitud/respuesta finalizado en middleware');
    } else {
        // Llamar a la siguiente función de middleware en la pila
        next();
    }
}
```

Archivos estáticos

Para servir archivos estáticos en una aplicación Express, se utiliza el middleware express.static. Este middleware permite servir archivos como imágenes, CSS, JavaScript y otros desde un directorio específico.

Pasos para server archivos estáticos:

- Configurar un servidor básico con Express.
- Configurar el middleware express.static para servir archivos desde un directorio.
- Crear algunos archivos estáticos (HTML, CSS, JS) en un directorio específico.
- Iniciar el servidor y probar la configuración.

Ejemplo:

- Estructura de la carpeta

```
/myapp
  /public
    /css
      styles.css
    /js
      scripts.js
```

```
    index.html
  server.js
```

- Crear el archivo server.js:

```
// Archivo: server.js

const express = require('express');
const path = require('path');
const app = express();
const PORT = 3000;

// Middleware para servir archivos estáticos desde el directorio 'public'
app.use(express.static(path.join(__dirname, 'public')));

// Ruta principal que redirige a index.html
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

// Iniciar el servidor
app.listen(PORT, () => {
    console.log(`Servidor escuchando en http://localhost:${PORT}`);
});
```

- Crear el archivo index.html dentro del directorio public.
- Crear el archivo styles.css dentro del directorio public/css.
- Crear el archivo scripts.js dentro del directorio public/js.
- Configurar el middleware express.static:

```
app.use(express.static(path.join(__dirname, 'public')));
```

Este middleware configura Express para servir archivos estáticos desde el directorio public. Cualquier archivo dentro de este directorio será accesible desde el navegador.

- Ruta principal que redirige a index.html:

```
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
```

Esta ruta sirve el archivo index.html cuando el usuario visita la ruta raíz.

JSON Web Tokens (JWT)

Los JSON Web Tokens (JWT) son un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente.

Componentes de un JWT

Un JWT consta de tres partes separadas por puntos ('.'):

- **Header** (Cabecera)
- **Payload** (Carga útil)
- **Signature** (Firma)

```
header.payload.signature
```

Header (Cabecera)

La cabecera generalmente consta de dos partes: el tipo de token, que es JWT, y el algoritmo de firma que se está utilizando, como HMAC SHA256 o RSA.

```
{
    "alg": "HS256",
    "typ": "JWT"
  }
```

Payload (Carga útil)

La carga útil es la parte del token que contiene las afirmaciones (claims). Las afirmaciones son declaraciones sobre una entidad (generalmente, el usuario) y datos adicionales. Existen tres tipos de afirmaciones:

- **Registered claims** (Afirmaciones registradas): un conjunto de afirmaciones predefinidas que no son obligatorias pero se recomiendan para proporcionar un conjunto de información útil, interoperable. Algunas son iss (emisor), exp (expiración), sub (sujeto), y aud (audiencia).
- **Public claims** (Afirmaciones públicas): estas pueden ser definidas libremente por aquellos que usan JWT. Para evitar colisiones, estas deben ser definidas en el registro de JWT de IANA o ser definidas como un nombre completamente calificado por URI.
- **Private claims** (Afirmaciones privadas): las que se crean para compartir información entre partes que han acordado usar esas afirmaciones y no están registradas o son públicas.

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
  }
```

3. Signature (Firma)

Para crear la firma, se toma el encabezado codificado en base64, la carga útil codificada en base64, un secreto, el algoritmo especificado en el encabezado y se firma esa cadena.

Ejemplo de firma usando HMAC SHA256:

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret)
```

La firma asegura que el mensaje no fue cambiado en el camino y, en el caso de los tokens firmados con una clave privada, puede también verificar que el remitente del JWT es quien dice ser.

Usar JSON Web Tokens (JWT) ofrece varias ventajas para la autenticación y la transmisión segura de información entre partes. Algunas razones clave para usar JWT:

Compacto

Los JWT son compactos y eficientes en cuanto a tamaño, ya que son representados como cadenas de caracteres en formato JSON, lo que facilita su transmisión a través de URLs, encabezados HTTP y otros medios de transporte.

Autónomo

Un JWT contiene toda la información necesaria sobre el usuario y sus permisos. Esto significa que no se requiere acceder a una base de datos en cada solicitud para verificar la autenticidad del token, lo que mejora el rendimiento y la escalabilidad de la aplicación.

Seguro

Los JWT pueden ser firmados utilizando un algoritmo de cifrado como HMAC SHA256 o RSA, lo que garantiza que los datos no hayan sido alterados. Además, pueden ser encriptados para agregar una capa adicional de seguridad.

Versátil

Los JWT se pueden utilizar en diversos contextos, como autenticación, autorización, intercambio de información segura, etc. Son agnósticos a los lenguajes de programación, lo que los hace útiles en arquitecturas de microservicios donde diferentes servicios pueden estar implementados en diferentes lenguajes.

Fácil de usar

Las bibliotecas para manejar JWT están disponibles en la mayoría de los lenguajes de programación, lo que facilita su implementación y manejo en cualquier stack tecnológico.

Ejemplo de Uso de JWT en Autenticación

A continuación, se muestra un ejemplo de cómo usar JWT en un sistema de autenticación en una aplicación Express:

Pasos a seguir:

- Configurar un servidor básico con Express.
- Implementar el inicio de sesión para generar un JWT.
- Crear un middleware para autenticar solicitudes usando JWT.
- Proteger rutas específicas utilizando el middleware de autenticación.

Código:

```
// Archivo: server.js

const express = require('express');
const jwt = require('jsonwebtoken');
const app = express();
const PORT = 3000;

const SECRET_KEY = 'your-256-bit-secret';

// Middleware para analizar cuerpos de solicitud JSON
app.use(express.json());

// Ruta de inicio de sesión para generar un JWT
app.post('/login', (req, res) => {
    const { username, password } = req.body;

    // Validar las credenciales del usuario
    if (username === 'user' && password === 'password') {
        // Crear el payload del JWT
        const payload = {
            sub: username,
            name: 'John Doe',
            admin: true
        };

        // Firmar el JWT
        const token = jwt.sign(payload, SECRET_KEY, { expiresIn: '1h' });

        res.json({ token });
    } else {
        res.status(401).send('Credenciales incorrectas');
    }
});
```

```
// Middleware para verificar el JWT
const authenticateJWT = (req, res, next) => {
    const token = req.headers.authorization;

    if (authHeader)

        jwt.verify(token, SECRET_KEY, (err, user) => {
            if (err) {
                return res.sendStatus(403);
            }
            req.user = user;
            next();
        });
    } else {
        res.sendStatus(401);
    }
};

// Ruta protegida
app.get('/protected', authenticateJWT, (req, res) => {
    res.send('Este es un recurso protegido');
});

// Iniciar el servidor
app.listen(PORT, () => {
    console.log(`Servidor escuchando en http://localhost:${PORT}`);
});
```