AGO
12

SPRINT 3 - MEETING 42

Servidor web con Express II

Grabación

Presentación

El intermediario entre las solicitudes de los/as usuarios/as y el servidor

> ***"Los servidores web tienen que estar protegidos frente a cualquier tipo de amenazas, tienen que estar preparados para ser el primer punto de entrada a cualquier compañía y, sobre todo, tienen que estar bien securizados." —Álvaro Roldán Peral. Fortinet Business Unit Manager ALTIMATE GROUP.***

Photo by Lilly Rum on Unsplash

En la bitácora anterior conociste el framework Express JS y utilizaste algunas de sus funciones para crear un servidor y determinar las rutas de acceso. En esta bitácora comenzaremos a pensar en la seguridad de las APIs que diseñamos, un factor esencial para proteger la información de los/as usuarios/as. ¡Para eso es hora de que conozcas la función middleware de Express JS!

Según Wikipedia, **middleware** o **lógica de intercambio de información entre aplicaciones** (interlogical) es:

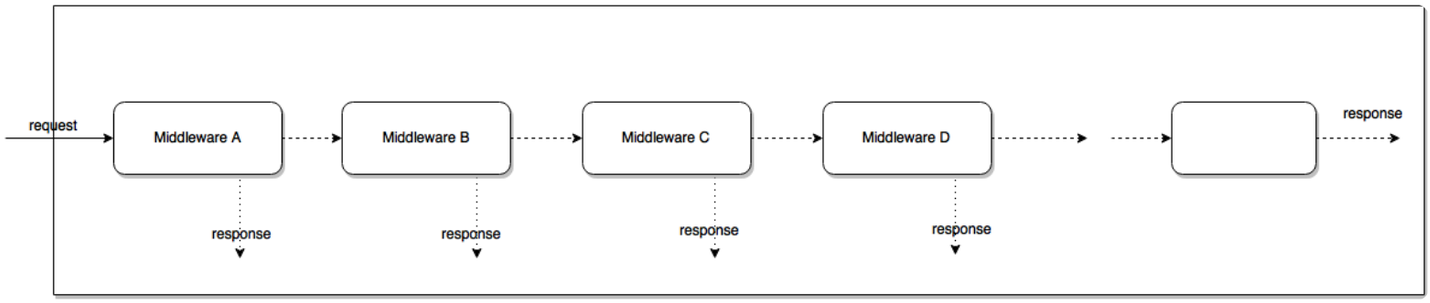
“un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware o sistemas operativos. Este simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones y sincronizaciones que son

calidad de servicio, así como la seguridad, el envío de mensajes, la actualización del directorio de servicio, etc.”

Además, al tratarse de una comunicación entre hardware y software, middleware tiene diferente usos. Tal como señala Agoi Abel (Javascript React NodeJs ExpressJs, PHP Laravel, *wannabe* blockchain developer) en este artículo:

“Un servidor web puede verse como una función que acepta una solicitud y genera una respuesta. ***Middlewares son funciones ejecutadas en el medio después de que la solicitud entrante produce una salida*** *que podría ser la salida final aprobada o podría ser utilizada por el siguiente middleware hasta que se complete el ciclo, lo que significa que podemos tener más de un middleware y se ejecutarán en el orden son declarados middleware A a continuación se ejecutará antes middleware B, middleware Bantes de middleware C.que podamos pasar variables de un middleware a otro.”*

Una imagen de este proceso podría verse así:



Fuente: “Una simple explicación de Middleware Express”

Esto puede suceder, por ejemplo, en el registro, la autorización o la gestión del estado de la sesión de un/a usuario/a. W. Julian en su artículo da un ejemplo de esto:

“*Imagine que su aplicación de alguna manera administra entidades de restaurantes. Solo los propietarios de restaurantes podrán cambiar o eliminar un restaurante. Por lo tanto, ambos puntos finales deberán verificar que una solicitud esté autenticada. Las funciones de middleware son un buen enfoque para lograr este comportamiento de forma segura y legible.”*

Los middleware son una forma simplificada de administrar diferentes rutas. Se encuentran detrás del procesamiento y enrutamiento de solicitudes. Es decir, en Express JS **los middlewares definen la secuencia de pasos de procesamiento de una solicitud en el servidor.** Es recomendable utilizarla cuando desarrollas una aplicación, ya que será quien reciba las solicitudes antes de que lleguen a las rutas y activará varias funciones.

Algunas de las tareas que puede realizar son:

- Ejecutar una pequeña porción de código.

- Finalizar el ciclo de solicitud-respuesta enviando respuesta.
- Llamar al siguiente middleware en la secuencia usando next ().

Puedes encontrar la lista completa en la página oficial.

Funcionamiento de los Middleware

Existen dos tipos de Middleware: los **globales** y los **particulares.**

Middlewares globales. Son aquellos que se ejecutarán cada vez que una ruta reciba algún request.

Middlewares particulares. Son aquellos que voy a asignar a una o varias rutas en particular.

Ambos trabajan de la misma manera: son ejecutados antes de la ruta y allí tenemos la potestad de frenar el request retornando un error y que nuestra ruta nunca se ejecute, o bien, continuar la ejecución a través del método `next()`.

Para definir un Middleware global, es preciso ejecutar el método use en la variable donde esté el `express()` instanciado.

```
app.use(function(req, res, next){
    console.log('Ejecutando Middleware');
    if(valido algo){
        next();
    } else {
        res.json({msj: ‘ha ocurrido un error’})
    }
})
```

Cada vez que una ruta intente ser ejecutada se llama al middleware global que validará la o las condiciones que se necesiten y dejará continuar con el flujo o responderá con un error.

Ahora bien, supongamos que tenemos a nuestro servidor escuchando peticiones de 10 rutas diferentes y solo quieres aplicar un middleware a solo 3 de ellas. Con el middleware global lo estarías ejecutando para las 10 rutas, pero a través de middlewares particulares podrás especificar qué rutas ejecutar.

Solo debes definir una función y pasarla como segundo parámetro en la definición de tu ruta, tal como muestra el siguiente ejemplo:

```
let particular = function(req, res, next){
        console.log('Particular');
        next();
}
```

```
app.get('/mi_ruta', particular, function (req, res) {
    res.send('Mi ruta');
});
```

En el siguiente ejemplo vamos a crear un server con dos rutas: un middleware global para ambas rutas y uno particular para una de ellas. En los resultados podrás ver fácilmente cómo se comportan:

```
var express = require('express');
var app = express();

app.use(function(req, res, next){
        console.log('Middle');
        next();
})

let particular = function(req, res, next){
        console.log('Particular');
        next();
}

app.get('/mi_ruta', particular, function (req, res) {
    res.send('Mi ruta');
});

app.get('/otra_ruta', function (req, res) {
        res.send('Mi otra ruta');
});
```

Cuando ejecutes “mi_ruta” podrás ver la salida de tu consola:

`Middle`

`Particular`

Cuando ejecutes “otra_ruta” podrás ver la salida de tu consola solamente:

`Middle`

Como se puede ver, el enrutamiento es muy cómodo para este tipo de funciones. Se puede utilizar también para identificar errores, identificar usuarios o hacer ciertas peticiones de forma genérica y múltiples endpoint.

Prestar atención a los errores

Durante el desarrollo de nuestros servicios, es importante pensar en los casos típicos en los que el cliente o el servidor suele tener algún problema.

Al diseñar una API con Express, deberías considerar, por ejemplo, qué ocurre si el cliente no envía toda la información requerida; o si hay algún problema en el servidor -quizás una variable no definida-; o bien qué pasará si hay problemas con la conexión a la base de datos. Uno de los usos comunes de las funciones middleware, de la que podrás beneficiarte, es el **manejo de errores** en la aplicación.

Un error común podría ser el de un recurso no encontrado, comúnmente conocido como “error 404”. Para retornar al usuario esta respuesta, en express se utiliza el res.status(404), que va a retornar dentro de los headers el código de estado 404. También podría enviar información a seguir del código de estado, por ejemplo res.status(404).send('Artículo no encontrado'). Veamos este ejemplo en una ruta de express:

```
const articulos = [
  {
    id: 1,
    titulo: 'Lorem ipsum',
  }, {
    id: 2,
    titulo: 'Donec tincidunt vestibulum magna',
  },
]

app.get('/articulo/:id', (req, res) => {
  const articulo = articulos.find(item => item.id ===
req.params.id);
  if (!articulo) {
    return res.status(404).send('Artículo no encontrado');
  }
  res.send(articulo);
 });
```

En este ejemplo, agregamos un condicional para que, en el caso de que el formulario de contacto enviado por el/la usuario/a esté incompleto, el servidor pueda retornar un error 400

```
app.post('/contacto', (req, res) => {
  const {
    nombre,
    email,
  } = req.body;

  if (!nombre || !email) {
    res.status(400);
    res.json({ error: 'Faltan datos obligatorios' });
    return;
  }

  // Continua ejecucion del codigo

});
```

Por último, exploraremos cómo utilizar un middleware para manejar los errores genéricos de express. En este caso, el middleware debería estar definido al final de nuestras rutas de express para asegurar que sea el último en ser ejecutado. A diferencia de los middlewares que vimos anteriormente, este tiene 4 parámetros en vez de 3:

```
app.use(function(err, req, res, next) {
    if(!err) return next();
    console.log('Error, algo salio mal', err);
    res.status(500).send('Error');
});
```

¡Recuerda consultar la documentación oficial de Express y el listado de *status codes* disponibles!

En resumen

Ya sabemos qué sucede detrás de una aplicación Express JS. Cuando recibe una solicitud se activan varias funciones denominadas middleware. Es muy importante saber cómo funciona el proceso antes de diseñar una aplicación, ¡la seguridad de los datos de los/as usuarios/as es un aspecto absolutamente relevante! Ya tienes todo lo que necesitas para crear tus propias APIs, ¿qué estás esperando?

¡Prepárate para el próximo encuentro!

Profundiza

Te invitamos a conocer más sobre el tema de esta bitácora.

Challenge

Te proponemos el siguiente desafío, ¿te animas?